

Formal Semantics and Soundness of a Translation from Event-B Actions to SQL Statements

Tim Wahls

Department of Mathematics and Computer Science
Dickinson College
wahlst@dickinson.edu

June 9, 2016

Abstract

The EventB2SQL tool translates Event-B models to persistent Java applications that store the state of the model in a relational database. Most Event-B assignments are translated directly to SQL database modification statements, which can then be executed against the database. In this work, we present a formal semantics for and prove the soundness of the translation of sets of assignment statements representing the actions of an Event-B event. This allows the generated code to be used with confidence in its correctness.

1 Introduction

The EventB2SQL tool¹ [21] translates Event-B [2] models to Java applications that store the state of the model in a relational database (either MySQL or SQLite). This makes the application persistent. Each event is translated as a database transaction, so the generated code is readily used in multi-threaded and client-server applications. Carrier sets are translated as Java generic type parameters and elements of those sets are stored in the database, allowing the code generated from the model to manipulate almost any Java object. EventB2SQL has been used in the re-development of an Enterprise Resource Planning system [6] and the development of an Android application for checking medication interactions and contraindications [20].

Soundness is particularly critical for code generators for Event-B, as Event-B models are typically verified to satisfy correctness and safety properties. If the code generated from such models does not satisfy those same properties, the code generation tool is worse than useless and may even present a danger to persons or property. Most Event-B code generators [16, 7, 8, 22] translate concrete models that are already relatively close to code, so that the correspondence between

¹<https://sourceforge.net/projects/eventb2sql/>

model and code is relatively straightforward. This may partially explain why no proof of the the soundness of the translation performed by most of these tools has been done. The refinement proofs performed as the abstract model is refined to a concrete one suitable for translation in some sense stand in for a soundness proof for the code generation algorithm. EventB2SQL is different in that it is meant to translate relatively abstract Event-B machines that use features such as simultaneous assignment, quantifiers, set comprehensions and variables of set and relation types. Hence, the conceptual distance between the translation source and target is much greater, and the correctness of that translation much less readily apparent. Refinement chains when using EventB2SQL are short or even non-existent. This is a tremendous advantage in terms of the time and effort needed to reach executable code, but the relationship between the abstract model and final implementation is much less apparent to the user. Hence, assuring the user of the soundness of the code generated by EventB2SQL is critical.

EventB2SQL translates most Event-B assignment statements directly to SQL `insert` and `delete` statements, which are then executed against the model state as represented in the database. In this work, we give a formal semantics for and prove the soundness of this part of the translation, thus allowing the code generated by EventB2SQL to be used with confidence in its correctness. We first give formal semantics for SQL and Event-B, as adapted from appropriate sources. We then show the soundness of the translation of Event-B expressions to SQL queries (Section 3.1), and finally the soundness of the translation of a set of Event-B assignment statements (representing the body of an event) to SQL `insert` and `delete` statements (Section 3.2).

2 Semantic Definitions

2.1 Formal Semantics of SQL

We construct a formal semantics for SQL by combining the translation of SQL queries to Entity-Relationship Calculus (ERC) given in [11] with the denotational-style semantics of `insert` and `delete` statements given in [15]. Following [15], we define a database db as a function from relation names to relations. A relation is a set of tuples with named attributes. We then modify the `sql2erc` operator of [11] from a syntax transformer to a semantic function that takes an SQL query and a database as its arguments, and returns a relation as its result (Figure 1). When the returned relation has a single tuple and a single attribute, we often treat it as an atomic (primitive) value. In the semantics, r is a relation name, τ_i represents a term, $d_i\#$ the set of all values in the database of the type of term τ_i , and φ_i a boolean valued formula. s_i is a tuple variable ranging over relation valued expressions r_i , and tuple variables are unique within a query. A is an attribute, and $A(s)$ is the value of attribute A for tuple variable s . Finally, $:$ is used for set membership, and ω is a binary operator. The semantics assumes that no tables contain duplicate elements, and that no queries produce

$$\begin{aligned}
\text{sql2erc}[\mathbf{r}, db] &= db(\mathbf{r}) \\
\text{sql2erc}[\text{select } \tau_1, \dots, \tau_n \\ &\quad \text{from } r_1 s_1, \dots, r_n s_n \\ &\quad \text{where } \varphi, db] &= \{ \text{res}_1, \dots, \text{res}_n \mid \text{res}_1 : d_1 \# \wedge \dots \wedge \text{res}_n : d_n \# \wedge \\ &\quad \exists s_1 : \text{sql2erc}[r_1, db] \cdot \dots \exists s_m : \text{sql2erc}[r_m, db] \cdot \\ &\quad \text{res}_1 = \text{sql2erc}[\tau_1, db] \wedge \dots \wedge \\ &\quad \text{res}_n = \text{sql2erc}[\tau_n, db] \wedge \text{sql2erc}[\varphi, db] \} \\
\text{sql2erc}[s.A, db] &= A(s) \\
\text{sql2erc}[\text{not } \varphi, db] &= (\neg \text{sql2erc}[\varphi, db]) \\
\text{sql2erc}[\varphi_1 \text{ and } \varphi_2, db] &= \text{sql2erc}[\varphi_1, db] \wedge \text{sql2erc}[\varphi_2, db] \\
\text{sql2erc}[\varphi_1 \text{ or } \varphi_2, db] &= \text{sql2erc}[\varphi_1, db] \vee \text{sql2erc}[\varphi_2, db] \\
\text{sql2erc}[\tau_1 \omega \tau_2, db] &= \text{sql2erc}[\tau_1, db] \omega \text{sql2erc}[\tau_2, db] \\
\text{sql2erc}[\tau \text{ in } (\text{select } s_i.A \\ &\quad \text{from } r_1 s_1, \dots, r_m s_m \\ &\quad \text{where } \varphi), db] &= \exists s_i : \text{sql2erc}[r_i, db] \cdot ((\exists s_1 : \text{sql2erc}[r_1, db] \dots \\ &\quad \exists s_{i-1} : \text{sql2erc}[r_{i-1}, db] \exists s_{i+1} : \text{sql2erc}[r_{i+1}, db] \dots \\ &\quad \exists s_m : \text{EB2SQL}[r_m, db]) \cdot \text{sql2erc}[\varphi, db]) \\ &\quad \wedge \text{sql2erc}[\tau = s_i.A, db] \\
\text{sql2erc}[\text{select } \tau_1, \dots, \tau_n \\ &\quad \text{from } r_1 s_1, \dots, r_n s_n \\ &\quad \text{where } \varphi \\ \text{union select } \tau'_1, \dots, \tau'_n \\ &\quad \text{from } r'_1 s'_1, \dots, r'_n s'_n \\ &\quad \text{where } \varphi', db] &= \{ \text{res}_1, \dots, \text{res}_n \mid \text{res}_1 : d_1 \# \wedge \dots \wedge \text{res}_n : d_n \# \wedge \\ &\quad \exists s_1 : \text{sql2erc}[r_1, db] \cdot \dots \exists s_m : \text{sql2erc}[r_m, db] \cdot \\ &\quad \text{res}_1 = \text{sql2erc}[\tau_1, db] \wedge \dots \wedge \text{res}_n = \text{sql2erc}[\tau_n, db] \wedge \\ &\quad \text{sql2erc}[\varphi, db] \\ &\quad \vee \exists s'_1 : \text{sql2erc}[r'_1, db] \cdot \dots \exists s'_m : \text{sql2erc}[r'_m, db] \cdot \\ &\quad \text{res}_1 = \text{sql2erc}[\tau'_1, db] \wedge \dots \wedge \text{res}_n = \text{sql2erc}[\tau'_n, db] \wedge \\ &\quad \text{sql2erc}[\varphi', db] \} \\
\text{sql2erc}[\text{select count}(s.A) \text{ from } rs, db] &= CNT\{A(s) \mid s : \text{sql2erc}[r, db]\}
\end{aligned}$$

Figure 1: The definition of the sql2erc operator.

$$\begin{aligned}
\text{sql2erc}_A[\text{insert ignore into } r \ S, db] &= [r \rightarrow db(r) \cup \text{sql2erc}[S, db]] db \\
\text{sql2erc}_A[\text{delete from } r, db] &= [r \rightarrow \{\}] db \\
\text{sql2erc}_A[\text{delete from } r \text{ where } \varphi, db] &= [r \rightarrow db(r) \setminus \\
&\quad \text{sql2erc}[\text{select } rtmp.A_1, \dots, rtmp.A_n \\
&\quad \text{from } r \ rtmp \\
&\quad \text{where } [rtmp/r]\varphi, db]] db \\
\text{sql2erc}_A[\lambda, db] &= db \\
\text{sql2erc}_A[A; AS, db] &= \text{sql2erc}_A[AS, \text{sql2erc}_A[A, db]]
\end{aligned}$$

Figure 2: The definition of the sql2erc_A operator.

duplicates. All queries produced by EventB2SQL that could potentially have duplicates in their results use the keyword `distinct` to eliminate such duplicates. Where needed, justification that queries do not produce duplicates is provided in the following proofs.

Beyond the adaptation to a denotational style, the definition of sql2erc given in Figure 1 differs from the definition given in [11] in two significant ways:

- the sql2erc operator is applied to relation valued expressions r_i appearing in the **from** clause of an SQL query. This was not done in [11], so the semantics given there does not consider derived relations (subqueries nested within the **from** clause of another query). This may be due to the fact that derived relations were a relatively new feature in SQL at the time of that work.
- the rule for count is simplified from the rule for the sql2erc_γ operator that translates queries with **group by** and **having** clauses

Next, we introduce a new semantic function sql2erc_A that defines the semantics of **insert** and **delete** statements. sql2erc_A takes one or more of these statements and a database as its arguments, and returns an updated database as its result. The definitions in Figure 2 are adapted from the definitions of semantic functions *Value_InSel*, *Value_Del* and *Value_DelCond* in [15]. In the first rule, S is an SQL query with a return value that is compatible with relation r (in terms of number of attributes and attribute types). In the second rule for **delete**, A_1, \dots, A_n are all of the attributes of r . The notation $[r \rightarrow v]db$ is used for updating functions, and evaluates to a function that is the same as db except that r maps to v . The notation $[rtmp/r]\varphi$ evaluates to φ with all free occurrences of r replaced by $rtmp$. This is done for compatibility with the notation used in [11]. The final two rules gives the semantics of a sequence of **insert** and **delete** statements, where λ is an empty sequence, A represents a single statement, and AS a sequence of statements. The final rule evaluates AS in the state resulting from evaluating A .

$$\begin{aligned}
\text{eb}[\![v, m]\!] &= m(v) \\
\text{eb}[\![\neg b, m]\!] &= \neg \text{eb}[\![b, m]\!] \\
\text{eb}[\![b_1 \wedge b_2, m]\!] &= \text{eb}[\![b_1, m]\!] \wedge \text{eb}[\![b_2, m]\!] \\
\text{eb}[\![b_1 \vee b_2, m]\!] &= \text{eb}[\![b_1, m]\!] \vee \text{eb}[\![b_2, m]\!] \\
\text{eb}[\![x \in s, m]\!] &= \text{eb}[\![x, m]\!] \in \text{eb}[\![s, m]\!] \\
\text{eb}[\![x \subset s, m]\!] &= \text{eb}[\![x, m]\!] \subset \text{eb}[\![s, m]\!] \\
\text{eb}[\![x \subseteq s, m]\!] &= \text{eb}[\![x, m]\!] \subseteq \text{eb}[\![s, m]\!] \\
\text{eb}[\![s_1 \cup s_2, m]\!] &= \{x \mid x \in \text{eb}[\![s_1, m]\!] \vee x \in \text{eb}[\![s_2, m]\!]\} \\
\text{eb}[\![s_1 \cap s_2, m]\!] &= \{x \mid x \in \text{eb}[\![s_1, m]\!] \wedge x \in \text{eb}[\![s_2, m]\!]\} \\
\text{eb}[\![s_1 \setminus s_2, m]\!] &= \{x \mid x \in \text{eb}[\![s_1, m]\!] \wedge x \notin \text{eb}[\![s_2, m]\!]\} \\
\text{eb}[\![\text{card}(s), m]\!] &= |\text{eb}[\![s, m]\!]| \\
\text{eb}[\![s_1 \times s_2, m]\!] &= \{x \mapsto y \mid x \in \text{eb}[\![s_1, m]\!] \wedge y \in \text{eb}[\![s_2, m]\!]\} \\
\text{eb}[\![\text{dom}(r), m]\!] &= \{x \mid \exists y. x \mapsto y \in \text{eb}[\![r, m]\!]\} \\
\text{eb}[\![\text{ran}(r), m]\!] &= \{y \mid \exists x. x \mapsto y \in \text{eb}[\![r, m]\!]\} \\
\text{eb}[\![s \triangleleft r, m]\!] &= \{x \mapsto y \mid x \mapsto y \in \text{eb}[\![r, m]\!] \wedge x \in \text{eb}[\![s, m]\!]\} \\
\text{eb}[\![s \triangleleft r, m]\!] &= \{x \mapsto y \mid x \mapsto y \in \text{eb}[\![r, m]\!] \wedge x \notin \text{eb}[\![s, m]\!]\} \\
\text{eb}[\![r \triangleright s, m]\!] &= \{x \mapsto y \mid x \mapsto y \in \text{eb}[\![r, m]\!] \wedge y \in \text{eb}[\![s, m]\!]\} \\
\text{eb}[\![r \triangleright s, m]\!] &= \{x \mapsto y \mid x \mapsto y \in \text{eb}[\![r, m]\!] \wedge y \notin \text{eb}[\![s, m]\!]\} \\
\text{eb}[\![r_1; r_2, m]\!] &= \{x \mapsto y \mid \exists z. x \mapsto z \in \text{eb}[\![r_1, m]\!] \wedge z \mapsto y \in \text{eb}[\![r_2, m]\!]\} \\
\text{eb}[\![r_1 \circ r_2, m]\!] &= \text{eb}[\![r_2; r_1, m]\!] \\
\text{eb}[\![r_1 \triangleleft r_2, m]\!] &= \text{eb}[\![r_2 \cup (\text{dom}(r_2) \triangleleft r_1), m]\!] \\
\text{eb}[\![r^{-1}, m]\!] &= \{y \mapsto x \mid x \mapsto y \in \text{eb}[\![r, m]\!]\} \\
\text{eb}[\![r[s], m]\!] &= \{y \mid \exists x. x \in \text{eb}[\![s, m]\!] \wedge x \mapsto y \in \text{eb}[\![r, m]\!]\} \\
\\
\text{eb}_A[\![v := E, m]\!] &= [v \rightarrow \text{eb}[\![E, m]\!]]m \\
\\
\text{eb}_{AS}[\![\lambda, m]\!] &= m \\
\text{eb}_{AS}[\![v := E \mid AS, m]\!] &= [v \rightarrow \text{eb}[\![E, m]\!]]\text{eb}_{AS}[\![AS, m]\!]
\end{aligned}$$

Figure 3: The definition of the eb , eb_A and eb_{AS} operators.

2.2 Formal Semantics of Event-B

In the usual denotational style, we define the state of an Event-B model as a function m mapping Event-B machine variable names (identifiers) to values. The possible value types are integer, boolean, sets of integers and booleans, and sets of ordered pairs of integers and/or booleans. Figure 3 presents the operators **eb** (defining the semantics of an Event-B expression in a given state) and \mathbf{eb}_A (defining the state produced by an assignment). The definition of the **eb** operator is adapted from the semantics of Event-B expressions given in Sect. 3.3 (Mathematical Notation) of the Rodin User’s Handbook [19]. In the semantics, v is a machine variable name, b is a boolean expression, x, y are set elements, s, s_1, s_2 are set valued expressions, and r, r_1, r_2 are relation valued expressions. In Event-B, $s \triangleleft r$ returns the pairs in r with key occurring in s , $s \triangleleft r$ the pairs in r with key not occurring in s , $r \triangleright s$ the pairs in r with value occurring in s , and $r \triangleright s$ the pairs in r with value not occurring in s . The $;$ operator is relational composition, \circ is reverse composition, \triangleleft is relational overriding, $^{-1}$ relational inverse, and \square relational image of a set of domain elements. In the rule for \mathbf{eb}_A , E is an Event-B expression.

In the rules for \mathbf{eb}_{AS} , λ represents 0 assignment statements, E an Event-B expression, and AS a set of 0 or more assignments. Because Event-B uses simultaneous assignment, the right hand side of each assignment is executed in the initial state m . An event is allowed to assign each machine variable at most once, so the order in which the assignments are processed and new values are associated with machine variables does not matter.

2.3 Formal Semantics of the Translation

Figure 4 gives the definition of the EB2SQL operator, which translates from Event-B expressions and predicates to SQL queries. EB2SQL is defined inductively. It is a purely syntactic transformation, so no machine state is needed at this point. In the definition, s is an Event-B machine variable of a set type, r is a relation variable, b, b_1 and b_2 are Event-B predicates; x and y are set elements; and r, r_1 and r_2 are relation valued expressions; f is a function valued expression; and s, s_1 and s_2 are set valued expressions that are not relations. Sets are represented as database tables with a single column called `refkey`, while relations are represented as tables with two columns: `id` and `value`. Hence, an operation such as $\text{dom}(r)$ which is defined in Event-B as $\{x \mid \exists y \cdot x \mapsto y \in r\}$ becomes:

```
select distinct rtmp.id from EB2SQL( $r$ ) rtmp
```

in SQL. Tuple variables `stmp`, `s1tmp`, `s2tmp`, `rtmp`, `r1tmp` and `r2tmp` represent a fresh variable name in each rule application, so no scoping rules are needed.

The use of some basic set theory in the rules for the $=$, \subseteq , \subset and \in operators precisely captures the implementation of these operators in the EventB2SQL tool. MySQL does not define the ANSI-standard **intersect** and **except** operators, so the query for $s_1 \cap s_2$ selects precisely the elements of s_1 that also occur in s_2 , and for $s_1 \setminus s_2$ the elements of s_1 that do not occur in s_2 .

$\text{EB2SQL}(s)$	<code>= select stmp.refkey from s stmp</code>
$\text{EB2SQL}(r)$	<code>= select rtmp.id, rtmp.value from r rtmp</code>
$\text{EB2SQL}(b_1 \wedge b_2)$	<code>= EB2SQL(b_1) and EB2SQL(b_2)</code>
$\text{EB2SQL}(b_1 \vee b_2)$	<code>= EB2SQL(b_1) or EB2SQL(b_2)</code>
$\text{EB2SQL}(\neg b)$	<code>= not EB2SQL(b)</code>
$\text{EB2SQL}(x = y)$	<code>= EB2SQL(x) = EB2SQL(y)</code>
$\text{EB2SQL}(\text{card}(s))$	<code>= select count(stmp.refkey) from EB2SQL(s) stmp</code>
$\text{EB2SQL}(s_1 \cup s_2)$	<code>= select sltmp.refkey from EB2SQL(s_1) sltmp union select stmp.refkey from EB2SQL(s_2) stmp</code>
$\text{EB2SQL}(s_1 \cap s_2)$	<code>= select sltmp.refkey from EB2SQL(s_1) sltmp, EB2SQL(s_2) stmp where sltmp.refkey = stmp.refkey</code>
$\text{EB2SQL}(s_1 \times s_2)$	<code>= select sltmp.refkey, stmp.refkey from EB2SQL(s_1) sltmp, EB2SQL(s_2) stmp</code>
$\text{EB2SQL}(s_1 \setminus s_2)$	<code>= select sltmp.refkey from EB2SQL(s_1) sltmp where sltmp.refkey not in (select stmp.refkey from EB2SQL(s_2) stmp)</code>
$\text{EB2SQL}(s_1 = s_2)$	<code>= EB2SQL($s_1 \cap s_2 = s_1 \wedge s_1 = s_2$)</code>
$\text{EB2SQL}(s_1 \subseteq s_2)$	<code>= EB2SQL($s_1 \cap s_2 = s_1$)</code>
$\text{EB2SQL}(s_1 \subset s_2)$	<code>= EB2SQL($s_1 \cap s_2 = s_1 \wedge s_1 \neq s_2$)</code>
$\text{EB2SQL}(x \in s)$	<code>= EB2SQL($\{x\} \subseteq s$)</code>
$\text{EB2SQL}(\text{dom}(r))$	<code>= select distinct rtmp.id from EB2SQL(r) rtmp</code>
$\text{EB2SQL}(\text{ran}(r))$	<code>= select distinct rtmp.value from EB2SQL(r) rtmp</code>
$\text{EB2SQL}(s \triangleleft r)$	<code>= select rtmp.id, rtmp.value from EB2SQL(r) rtmp, EB2SQL(s) stmp where rtmp.id = stmp.refkey</code>
$\text{EB2SQL}(s \triangleleft r)$	<code>= select rtmp.id, rtmp.value from EB2SQL(r) rtmp where rtmp.id not in (select stmp.refkey from EB2SQL(s) stmp)</code>
$\text{EB2SQL}(r \triangleright s)$	<code>= select rtmp.id, rtmp.value from EB2SQL(r) rtmp, EB2SQL(s) stmp where rtmp.value = stmp.refkey</code>
$\text{EB2SQL}(r \triangleright s)$	<code>= select rtmp.id, rtmp.value from EB2SQL(r) rtmp where rtmp.value not in (select stmp.refkey from EB2SQL(s) stmp)</code>
$\text{EB2SQL}(r_1; r_2)$	<code>= select distinct r1tmp.id, r2tmp.value from EB2SQL(r_1) r1tmp, EB2SQL(r_2) r2tmp where r1tmp.value = r2tmp.id</code>
$\text{EB2SQL}(r_1 \circ r_2)$	<code>= EB2SQL($r_2; r_1$)</code>
$\text{EB2SQL}(r_1 \triangleleft r_2)$	<code>= EB2SQL($r_2 \cup (\text{dom}(r_2) \triangleleft r_1)$)</code>
$\text{EB2SQL}(r^{-1})$	<code>= select rtmp.value, rtmp.id from EB2SQL(r) rtmp</code>
$\text{EB2SQL}(r[s])$	<code>= select distinct rtmp.value from EB2SQL(r) rtmp, EB2SQL(s) stmp where rtmp.id = stmp.refkey</code>

Figure 4: The definition of the EB2SQL operator.

$EB2SQL_A(s := s \cup s_1)$	$=$	<code>insert ignore into s select stmp.refkey from s' stmp</code>
$EB2SQL_A(s := s \setminus s_1)$	$=$	<code>delete from s where s.refkey in (select sltmp.refkey from s' sltmp)</code>
$EB2SQL_A(s := s \cap s_1)$	$=$	<code>delete from s where s.refkey in (select sltmp.refkey from s' sltmp)</code>
$EB2SQL_A(r := r \Leftarrow r_1)$	$=$	<code>delete from r where r.id in (select rltmp.id from r' rltmp); insert ignore into r select r2tmp.id, r2tmp.value from r' r2tmp</code>
$EB2SQL_A(r := s_1 \Leftarrow r)$	$=$	<code>delete from r where r.id in (select stmp.refkey from r' stmp)</code>
$EB2SQL_A(r := s_1 \triangleleft r)$	$=$	<code>delete from r where r.id in (select stmp.refkey from r' stmp)</code>
$EB2SQL_A(r := r \triangleright s_1)$	$=$	<code>delete from r where r.value in (select stmp.refkey from r' stmp)</code>
$EB2SQL_A(r := r \triangleright s_1)$	$=$	<code>delete from r where r.value in (select stmp.refkey from r' stmp)</code>
$EB2SQL_A(s := s_1)$	$=$	<code>delete from s; insert ignore into s select sltmp.refkey from s' sltmp</code>
$EB2SQL_A(r := r_1)$	$=$	<code>delete from r; insert ignore into r select rltmp.id, rltmp.value from r' rltmp</code>
$EB2SQL_{AS}[\lambda, db]$	$=$	<code>db</code>
$EB2SQL_{AS}[v := E AS, db]$	$=$	<code>EB2SQL_{AS}[AS, [v' → undef]sql2erc_A[[EB2SQL_A(v := E), db]]]</code>
$EB2SQL_O[s := s \cup s_1, db]$	$=$	<code>[s' → sql2erc[[EB2SQL(s_1), db]]db</code>
$EB2SQL_O[s := s \setminus s_1, db]$	$=$	<code>[s' → sql2erc[[EB2SQL(s_1), db]]db</code>
$EB2SQL_O[s := s \cap s_1, db]$	$=$	<code>[s' → sql2erc[[EB2SQL(s \setminus s_1), db]]db</code>
$EB2SQL_O[r := r \Leftarrow r_1, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(r_1), db]]db</code>
$EB2SQL_O[r := s_1 \Leftarrow r, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(s_1), db]]db</code>
$EB2SQL_O[r := s_1 \triangleleft r, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(dom(r) \setminus s_1), db]]db</code>
$EB2SQL_O[r := r \triangleright s_1, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(s_1), db]]db</code>
$EB2SQL_O[r := r \triangleright s_1, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(ran(r) \setminus s_1), db]]db</code>
$EB2SQL_O[s := s_1, db]$	$=$	<code>[s' → sql2erc[[EB2SQL(s_1), db]]db</code>
$EB2SQL_O[r := r_1, db]$	$=$	<code>[r' → sql2erc[[EB2SQL(r_1), db]]db</code>
$EB2SQL_{OS}[\lambda, db]$	$=$	<code>db</code>
$EB2SQL_{OS}[A AS, db]$	$=$	<code>EB2SQL_{OS}[AS, EB2SQL_O[A, db]]</code>
$EB2SQL_{RES}[AS, db]$	$=$	<code>EB2SQL_{AS}[AS, EB2SQL_{OS}[AS, db]]</code>

Figure 5: The definition of the $EB2SQL_A$, $EB2SQL_{AS}$, $EB2SQL_O$, $EB2SQL_{OS}$ and $EB2SQL_{RES}$ operators.

The EB2SQL_A operation (Figure 5) translates Event-B assignments into SQL `insert` and `delete` statements. The translation is again purely syntactic, so no state parameter is required. In cases where multiple statements are needed to translate one assignment, they are separated by semicolons. `tmp` is assumed to be a fresh (unused) table name. The EB2SQL_{RES} operator (defined at the end of Figure 5) uses the sql2erc_O operator (also defined in Figure 5) to evaluate an expression derived from the right hand side of the assignment and bind that result to the primed identifier (either s' or r') in the state used to evaluate the query. The first eight rules capture the optimizations introduced in [6]. The ninth rule could be used for an assignment where any of the first three rules could be used, and the tenth rule could be used where any of rules four through eight could be used. This is an issue for efficiency only, not correctness, as we will prove soundness no matter which of the applicable rules is used. The implementation of the EventB2SQL tool always uses the first applicable rule. The EB2SQL_{AS} operation (also in Figure 5) translates a set of Event-B assignment statements into SQL. E is an expression of a set or relation type that is compatible with the type of v . As evaluating the result of applying EB2SQL_A defines a primed version of the name on the left hand side of the assignment in the resulting state, we use $[v' \rightarrow \text{undef}]$ to ensure that v' is not defined in the overall result of evaluating the collection of Event-B assignments. As shown in the proof of Theorem 2, this technique for evaluating a collection of assignments preserves the simultaneous assignment semantics of Event-B, so the order that the assignments are evaluated in does not matter.

The EB2SQL_O operator (still in Figure 5) binds a primed version of the name on the left hand side of an assignment to a value derived from the expression on the right hand side. This value is then used in evaluating the query that EB2SQL_A generates for the assignment. The effects of using EB2SQL_A and EB2SQL_O together are as follows:

- for $s := s \cup s_1$: insert all values in s_1 that do not already occur in s into s
- for $s := s \setminus s_1$: delete all values in s_1 from s
- for $s := s \cap s_1$: delete all values in $s \setminus s_1$ from s
- for $r := r \Leftarrow r_1$: delete all pairs in r with key occurring as a key in r_1 , then insert all pairs in r_1 into r
- for $r := s_1 \Leftarrow r$: delete all pairs in r with key occurring in s_1
- for $r := s_1 \Leftarrow r$: delete all pairs in r with key occurring in $\text{dom}(r) \setminus s_1$
- for $r := r \triangleright s_1$: delete all pairs in r with value occurring in s_1
- for $r := r \triangleright s_1$: delete all pairs in r with value occurring in $\text{ran}(r) \setminus s_1$
- for $s := s_1$: delete all values from s , then insert all values in s_1 into s
- for $r := r_1$: delete all pairs from r , then insert all pairs in r_1 into r

$$\begin{aligned}
rep_E(i) &= i \\
rep_E(b) &= b \\
rep_E(\{\}) &= \{\} \\
rep_E(\{\text{refkey: } x\} \cup S) &= \{x\} \cup rep_E(S) \\
rep_E(\{\text{id: } x, \text{value: } y\} \cup R) &= \{x \mapsto y\} \cup rep_E(R) \\
rep(db) &= \{i \mapsto rep_E(v) \mid i \rightarrow v \in db\}
\end{aligned}$$

Figure 6: The definition of the *rep* operator that maps database states to Event-B states.

Finally, $EB2SQL_{OS}$ returns the state resulting from applying $EB2SQL_O$ to each of a collection of assignments, and $EB2SQL_{RES}$ uses $EB2SQL_{AS}$ to evaluate a collection of assignments in the state resulting from applying $EB2SQL_{OS}$ to that same collection.

3 Proof of Soundness

To show the soundness of the translation, we first define a mapping function to relate database states to Event-B model states. Function *rep* (Figure 6) uses a helper function rep_E to translate SQL values to Event-B values. It removes attribute names from tuples and converts 1-tuples to integers or booleans. In Figure 6, i is an integer constant, b a boolean constant, S a table representing a set and R a table representing a relation or function. The role of *rep* is similar to that of gluing invariants in Event-B refinement.

With these mechanics in hand, we are finally in a position to state the soundness of the translation performed by the EventB2SQL tool as a pair of theorems. The first states the soundness of the translation of expressions performed by $EB2SQL$, and is presented in Section 3.1. The second states the soundness of the translation of collections of Event-B assignment statements, and is given in Section 3.2.

3.1 Proof of Soundness: Translating Expressions

Theorem 1. *For every Event-B expression E that is translated directly to an SQL query by the EventB2SQL tool and every correctly typed database state db :*

$$rep_E(sql2erc[EB2SQL(E), db]) = eb[E, rep(db)]$$

Hence, the SQL query is equivalent to the Event-B expression that it was translated from.

We prove Theorem 1 by structural induction. The induction takes the form of a case analysis, with one case for each rule for $EB2SQL$ in Figure 4. The second line of each proof case uses the rule, and in all but trivial cases the

second to last line gives the definition of the operator in Event-B from Sect. 3.3 (Mathematical Notation) of the Rodin User's Handbook [19]. At the base of the induction, an Event-B set (which may be a relation) is represented by a database table containing the elements of the set. As a reminder, relations are represented by tables with columns id and value, and other sets by tables with a single column called refkey. Predicates of the form $\text{res}_i : d_i \#$ are dropped in the proofs when it is established that res_i is a value in the database. We use *subs.* to justify a substitution of equals for equals, *I.H.* for an application of the inductive hypothesis, *ERC* a step by the definition of the Entity-Relationship calculus, and the name of an operator to justify a step by definition of that operator.

The definitions and proofs for set operations card , \cup , \cap and \setminus are easily extended for relations. In the relation case, the proofs would have exactly the same structure except that there would be two attributes in the select clause (id and value) rather than the single attribute (refkey) in the non-relation case. Additionally, comparisons of the form $\text{s1tmp.refkey} = \text{s2tmp.refkey}$ would become $\text{r1tmp.id} = \text{r2tmp.id}$ and $\text{r1tmp.value} = \text{r2tmp.value}$. We assume type compatibility of all operands in Event-B expressions, as this is checked by Rodin. Several of the less interesting cases are omitted in the interest of space.

Case 1. $\text{rep}_E(\text{sql2erc}[\text{EB2SQL}(s), db]) = \text{eb}[s, \text{rep}(db)]$ (*basis*)

Proof.

$$\begin{aligned}
& \text{rep}_E(\text{sql2erc}[\text{EB2SQL}(s), db]) \\
&= \text{rep}_E(\text{sql2erc}[\text{select stmp.refkey from s stmp}, db]) && \text{EB2SQL} \\
&= \text{rep}_E(\{\text{res}_1 \mid \text{res}_1 : d_1 \# \wedge \exists \text{stmp} : \text{sql2erc}[s, db]. \\
&\quad \text{res}_1 = \text{sql2erc}[\text{stmp.refkey}, db]\}) && \text{sql2erc} \\
&= \text{rep}_E(\{\text{res}_1 \mid \text{res}_1 : d_1 \# \wedge \exists \text{stmp} : db(s). \\
&\quad \text{res}_1 = \text{refkey}(\text{stmp})\}) && \text{sql2erc} \\
&= \text{rep}_E(\{\text{refkey}(\text{stmp}) \mid \text{stmp} : db(s)\}) && \text{subs.} \\
&= \{x \mid x \in \text{rep}(db)(s)\} && \text{rep}_E \\
&= \text{rep}(db)(s) && \text{rep} \\
&= \text{eb}[s, \text{rep}(db)] && \text{eb } \square
\end{aligned}$$

Case 2. $\text{rep}_E(\text{sql2erc}[\text{EB2SQL}(r), db]) \equiv \text{eb}[r, \text{rep}(db)]$ (*basis*)

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(r), db]) \\
&= rep_E(sql2erc[select rtmp.id, rtmp.value from r rtmp, db]) && \text{EB2SQL} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \wedge \exists rtmp : sql2erc[r, db] \cdot \\
&\quad res_1 = sql2erc[rtmp.id, db] \wedge res_2 = sql2erc[rtmp.value, db]\}) && \text{sql2erc} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \wedge \exists rtmp : db(r) \cdot \\
&\quad res_1 = id(rtmp) \wedge res_2 = value(rtmp)\}) && \text{sql2erc} \\
&= rep_E(\{id(rtmp), value(rtmp) \mid rtmp : db(r)\}) && \text{subs.} \\
&= \{x \mapsto y \mid x \mapsto y \in rep(db)(r)\} && rep_E \\
&= rep(db)(r) && rep \\
&= eb[r, rep(db)] && eb \quad \square
\end{aligned}$$

Case 3. $rep_E(sql2erc[EB2SQL(b_1 \wedge b_2), db]) \equiv eb[b_1 \wedge b_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(b_1 \wedge b_2), db]) \\
&\equiv rep_E(sql2erc[EB2SQL(b_1) \text{ and } EB2SQL(b_2), db]) && \text{EB2SQL} \\
&\equiv rep_E(sql2erc[EB2SQL(b_1), db] \wedge sql2erc[EB2SQL(b_2), db]) && \text{sql2erc} \\
&\equiv rep_E(sql2erc[EB2SQL(b_1), db]) \wedge rep_E(sql2erc[EB2SQL(b_2), db]) && rep_E \\
&\equiv eb[b_1, rep(db)] \wedge eb[b_2, rep(db)] && \text{I.H.} \\
&\equiv eb[b_1 \wedge b_2, rep(db)] && eb \quad \square
\end{aligned}$$

Case 4. $sql2erc[EB2SQL(b_1 \vee b_2), db] \equiv eb[b_1 \vee b_2, rep(db)]$

Proof. Symmetric with Case 3. \square

Case 5. $rep_E(sql2erc[EB2SQL(\neg b), db]) \equiv eb[\neg b, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(\neg b), db]) \\
&\equiv rep_E(sql2erc[not EB2SQL(b), db]) && \text{EB2SQL} \\
&\equiv rep_E(\neg sql2erc[EB2SQL(b), db]) && \text{sql2erc} \\
&\equiv \neg rep_E(sql2erc[EB2SQL(b), db]) && rep_E \\
&\equiv \neg eb[b, rep(db)] && \text{I.H.} \\
&\equiv eb[\neg b, rep(db)] && eb \quad \square
\end{aligned}$$

Case 6. $rep_E(sql2erc[EB2SQL(x = y), db]) \equiv eb[x = y, rep(db)]$

Proof. Symmetric with Case 3. □

Case 7. $rep_E(sql2erc[EB2SQL(card(s), db)]) = eb[card(s), rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(card(s), db)]) \\
&= rep_E(sql2erc[\text{select count}(stmp.refkey) \\
&\quad \text{from EB2SQL}(s) \text{ stmp}, db]) \quad \text{EB2SQL} \\
&= rep_E(CNT\{\text{refkey}(stmp) \mid \text{stmp} : sql2erc[EB2SQL}(s), db]\}) \quad \text{sql2erc} \\
&= CNT\{x \mid x \in rep_E(sql2erc[EB2SQL}(s), db])\} \quad rep_E \\
&= CNT\{x \mid x \in eb[s, rep(db)]\} \quad \text{I.H.} \\
&= CNTeb[s, rep(db)] \quad \text{subs.} \\
&= |eb[s, rep(db)]| \quad \text{ERC} \\
&= eb[card(s), rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that `count` and `count distinct` give the same result when applied to an operand that does not contain duplicates.

Case 8. $rep_E(sql2erc[EB2SQL(s_1 \cup s_2), db]) = eb[s_1 \cup s_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s_1 \cup s_2), db]) \\
&= rep_E(sql2erc[\text{select s1tmp.refkey} \\
&\quad \text{from EB2SQL}(s_1) \text{ s1tmp union} \\
&\quad \text{select s2tmp.refkey} \\
&\quad \text{from EB2SQL}(s_2) \text{ s2tmp}, db]) \quad \text{EB2SQL} \\
&= rep_E(\{\text{res}_1 \mid \text{res}_1 : d_1 \# \wedge (\\
&\quad \exists \text{s1tmp} : sql2erc[EB2SQL}(s_1), db] \cdot \\
&\quad \text{res}_1 = sql2erc[\text{s1tmp.refkey}, db] \\
&\quad \vee \exists \text{s2tmp} : sql2erc[EB2SQL}(s_2), db] \cdot \\
&\quad \text{res}_1 = sql2erc[\text{s2tmp.refkey}, db])\}) \quad \text{sql2erc} \\
&= \{x \mid x \in rep_E([EB2SQL}(s_1), db]) \vee x \in rep_E(sql2erc[EB2SQL}(s_2), db])\} \quad \text{subs, } rep_E \\
&= \{x \mid x \in eb[s_1, rep(db)] \vee x \in eb[s_2, rep(db)]\} \quad \text{I.H.} \\
&= eb[s_1 \cup s_2, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that the `union` operator in SQL removes duplicates by default.

Case 9. $rep_E(sql2erc[EB2SQL(s_1 \cap s_2), db]) = eb[s_1 \cap s_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s_1 \cap s_2), db]) \\
&= rep_E(sql2erc[\text{select } s1tmp.refkey \\
&\quad \text{from } EB2SQL(s_1) \text{ } s1tmp, EB2SQL(s_2) \text{ } s2tmp \\
&\quad \text{where } s1tmp.refkey = s2tmp.refkey, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists s1tmp : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad \exists s2tmp : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad res_1 = sql2erc[s1tmp.refkey, db] \wedge \\
&\quad sql2erc[s1tmp.refkey = s2tmp.refkey, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists s1tmp : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad \exists s2tmp : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad res_1 = refkey(s1tmp) \wedge refkey(s1tmp) = refkey(s2tmp)\}) \quad \text{sql2erc} \\
&= \{x \mid x \in rep_E(sql2erc[EB2SQL(s_1), db]) \wedge \\
&\quad x \in rep_E(sql2erc[EB2SQL(s_2), db])\} \quad \text{subs., } rep_E \\
& \\
&= (\{x \mid x \in eb[s_1, rep(db)] \wedge x \in eb[s_2, rep(db)]\}) \quad \text{I.H.} \\
&= eb[s_1 \cap s_2, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that if s_1 and s_2 do not contain duplicates, then each element of s_1 matches at most one element of s_2 in the query above. This ensures that the query result does not contain duplicates. MySQL does not have the ANSI SQL standard `intersect` operator, so EB2SQL translates \cap as shown above.

Case 10. $rep_E(sql2erc[EB2SQL(s_1 \times s_2), db]) = eb[s_1 \times s_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s_1 \times s_2), db]) \\
&= rep_E(sql2erc[\text{select s1tmp.refkey, s2tmp.refkey} \\
&\quad \text{from EB2SQL}(s_1) \text{ s1tmp,} \\
&\quad \text{EB2SQL}(s_2) \text{ s2tmp, db}]) \quad \text{EB2SQL} \\
&= rep_E(\{\text{res}_1, \text{res}_2 \mid \text{res}_1 : d_1 \# \wedge \text{res}_2 : d_2 \# \wedge \\
&\quad \exists \text{s1tmp} : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad \exists \text{s2tmp} : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad \text{res}_1 = sql2erc[\text{s1tmp.refkey, db}] \wedge \\
&\quad \text{res}_2 = sql2erc[\text{s2tmp.refkey, db}]\}) \quad \text{sql2erc} \\
&= rep_E(\{\text{res}_1, \text{res}_2 \mid \text{res}_1 : d_1 \# \wedge \text{res}_2 : d_2 \# \wedge \\
&\quad \exists \text{s1tmp} : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad \exists \text{s2tmp} : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad \text{res}_1 = \text{refkey}(\text{s1tmp}) \wedge \text{res}_2 = \text{refkey}(\text{s2tmp})\}) \quad \text{sql2erc} \\
&= \{x \mapsto y \mid x \in rep_E(sql2erc[EB2SQL(s_1), db]) \wedge \\
&\quad y \in rep_E(sql2erc[EB2SQL(s_2), db])\} \quad \text{subs., } rep_E \\
&= \{x \mapsto y \mid x \in \text{eb}[s_1, rep(db)] \wedge y \in \text{eb}[s_2, rep(db)]\} \quad \text{I.H.} \\
&= \text{eb}[s_1 \times s_2, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that if s_1 and s_2 do not contain duplicates, the result of the query above can not contain duplicates.

Case 11. $rep_E(sql2erc[EB2SQL(s_1 \setminus s_2), db]) = \text{eb}[s_1 \setminus s_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s_1 \setminus s_2), db]) \\
&= rep_E(sql2erc[\text{select s1tmp.refkey} \\
&\quad \text{from EB2SQL}(s_1) \text{ s1tmp} \\
&\quad \text{where s1tmp.refkey not in (} \\
&\quad \quad \text{select s2tmp.refkey} \\
&\quad \quad \text{from EB2SQL}(s_2) \text{ s2tmp, db}]) \quad \text{EB2SQL}
\end{aligned}$$

$$\begin{aligned}
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists s1tmp : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad res_1 = sql2erc[s1tmp.refkey, db] \wedge \\
&\quad sql2erc[s1tmp.refkey \text{ not in } (\\
&\quad \quad \text{select s2tmp.refkey} \\
&\quad \quad \text{from EB2SQL}(s_2) \text{ s2tmp}), db]\}) \quad sql2erc \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists s1tmp : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad res_1 = refkey(s1tmp) \wedge \\
&\quad \neg \exists s2tmp : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad \quad sql2erc[s1tmp.refkey = s2tmp.refkey, db]\}) \quad sql2erc \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists s1tmp : sql2erc[EB2SQL(s_1), db] \cdot \\
&\quad res_1 = refkey(s1tmp) \wedge \\
&\quad \neg \exists s2tmp : sql2erc[EB2SQL(s_2), db] \cdot \\
&\quad \quad refkey(s1tmp) = refkey(s2tmp)\}) \quad sql2erc \\
&= \{x \mid x \in rep_E(sql2erc[EB2SQL(s_1), db]) \wedge \\
&\quad \neg x \in rep_E(sql2erc[EB2SQL(s_2), db])\} \quad \text{subs., } rep_E \\
&= \{x \mid x \in eb[EB2SQL(s_1), rep(db)] \wedge \\
&\quad x \notin eb[EB2SQL(s_2), rep(db)]\} \quad \text{I.H., eb} \\
&= eb[s_1 \setminus s_2, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that if s_1 does not contain duplicates, the result of the query above can not contain duplicates. MySQL does not have the ANSI SQL standard `except` operator, so `EB2SQL` translates `\` as shown above.

Case 12. $rep_E(sql2erc[EB2SQL(s_1 = s_2), db]) \equiv eb[s_1 = s_2, rep(db)]$

Proof. The proof is by Cases 3, 6, 7, and 9. \square

Case 13. $rep_E(sql2erc[EB2SQL(s_1 \subseteq s_2), db]) \equiv eb[s_1 \subseteq s_2, rep(db)]$

Proof. The proof is by Cases 6, 7, and 9. \square

Case 14. $rep_E(sql2erc[EB2SQL(s_1 \subset s_2), db]) \equiv eb[s_1 \subset s_2, rep(db)]$

Proof. The proof is by Cases 3, 5, 6, 7, and 9. \square

Case 15. $rep_E(sql2erc[EB2SQL(x \in s), db]) \equiv eb[x \in s, rep(db)]$

Proof. The proof follows Case 13. \square

Case 16. $rep_E(sql2erc[EB2SQL(dom(r)), db]) = eb[dom(r), rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(dom(r)), db]) \\
&= rep_E(sql2erc[\text{select distinct } rtmp.id \\
&\quad \text{from } EB2SQL(r) \text{ } rtmp, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad res_1 = sql2erc[rtmp.id, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad res_1 = id(rtmp)\}) \quad \text{sql2erc} \\
&= \{x \mid x \mapsto y \in rep_E(sql2erc[EB2SQL(r), db])\} \quad \text{subs., } rep_E \\
&= \{x \mid x \mapsto y \in eb[r, rep(db)]\} \quad \text{I.H.} \\
&= eb[dom(r), rep(db)] \quad \text{eb } \square
\end{aligned}$$

Case 17. $rep_E(sql2erc[EB2SQL(ran(r)), db]) = eb[ran(r), rep(db)]$

Proof. The proof is analogous with Case 16. \square

Case 18. $rep_E(sql2erc[EB2SQL(s \triangleleft r), db]) = eb[s \triangleleft r, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s \triangleleft r), db]) \\
&= rep_E(sql2erc[\text{select } rtmp.id, rtmp.value \\
&\quad \text{from } EB2SQL(r) \text{ } rtmp, EB2SQL(s) \text{ } stmp \\
&\quad \text{where } rtmp.id = stmp.refkey, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \exists stmp : sql2erc[EB2SQL(s), db] \cdot \\
&\quad res_1 = sql2erc[rtmp.id, db] \wedge res_2 = sql2erc[rtmp.value, db] \wedge \\
&\quad sql2erc[rtmp.id = stmp.refkey, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \exists stmp : sql2erc[EB2SQL(s), db] \cdot \\
&\quad res_1 = id(rtmp) \wedge res_2 = value(rtmp) \wedge \\
&\quad id(rtmp) = refkey(stmp)\}) \quad \text{sql2erc} \\
&= \{x \mapsto y \mid x \mapsto y \in rep_E(sql2erc[EB2SQL(r), db]) \\
&\quad \wedge x \in rep_E(sql2erc[EB2SQL(s), db])\} \quad \text{subs., } rep_E \\
&= \{x \mapsto y \mid x \mapsto y \in eb[r, rep(db)] \wedge x \in eb[s, rep(db)]\} \quad \text{I.H.} \\
&= eb[s \triangleleft r, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that if s does not contain duplicates, then the id of each tuple in r matches at most one element of s . Hence, if r does not contain duplicates, the result of the query above does not contain duplicates.

Case 19. $rep_E(sql2erc[EB2SQL(s \triangleleft r), db]) = eb[s \triangleleft r, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(s \triangleleft r), db]) \\
&= rep_E(sql2erc[\text{select } rtmp.id, rtmp.value \\
&\quad \text{from } EB2SQL(r) \text{ rtmp} \\
&\quad \text{where } rtmp.id \text{ not in} \\
&\quad \quad (\text{select } stmp.refkey \\
&\quad \quad \text{from } EB2SQL(s) \text{ stmp}), db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad res_1 = sql2erc[rtmp.id, db] \wedge \\
&\quad \quad res_2 = sql2erc[rtmp.value, db] \wedge \\
&\quad \quad sql2erc[rtmp.id \text{ not in} \\
&\quad \quad \quad (\text{select } stmp.refkey \\
&\quad \quad \quad \text{from } EB2SQL(s) \text{ stmp}), db]\} \quad \text{sql2erc} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad res_1 = id(rtmp) \wedge res_2 = value(rtmp) \wedge \\
&\quad \quad \neg \exists stmp : sql2erc[EB2SQL(s), db] \cdot \\
&\quad \quad \quad sql2erc[rtmp.id = stmp.refkey, db]\}) \quad \text{sql2erc} \\
&= \{x \mapsto y \mid x \mapsto y \in rep_E(sql2erc[EB2SQL(r), db]) \wedge \\
&\quad \quad x \notin rep_E(sql2erc[EB2SQL(s), db])\} \quad \text{subs., } rep_E \\
&= \{x \mapsto y \mid x \mapsto y \in eb[r, rep(db)] \wedge \\
&\quad \quad x \notin eb[s, rep(db)]\} \quad \text{I.H.} \\
&= eb[s \triangleleft r, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Note that if r does not contain duplicates, the result of the query above can not contain duplicates.

Case 20. $rep_E(sql2erc[EB2SQL(r \triangleright s), db]) = eb[r \triangleright s, rep(db)]$

Proof. The proof is analogous with Case 18. \square

Case 21. $rep_E(sql2erc[EB2SQL(r \triangleright s), db]) = eb[r \triangleright s, rep(db)]$

Proof. The proof is analogous with Case 19. \square

Case 22. $rep_E(sql2erc[EB2SQL(r_1 \triangleleft r_2), db]) = eb[r_1 \triangleleft r_2, rep(db)]$

Proof. The proof is by Cases 8, 16 and 19. \square

Case 23. $rep_E(sql2erc[EB2SQL(r_1; r_2), db]) = eb[r_1; r_2, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(r_1; r_2), db]) \\
&= rep_E(sql2erc[\text{select distinct } r1tmp.id, r2tmp.value \\
&\quad \text{from } EB2SQL(r_1) \text{ } r1tmp, EB2SQL(r_2) \text{ } r2tmp \\
&\quad \text{where } r1tmp.value = r2tmp.id, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists r1tmp : sql2erc[EB2SQL(r_1), db] \cdot \\
&\quad \exists r2tmp : sql2erc[EB2SQL(r_2), db] \cdot \\
&\quad res_1 = sql2erc[r1tmp.id, db] \wedge \\
&\quad res_2 = sql2erc[r2tmp.value, db] \wedge \\
&\quad sql2erc[r1tmp.value = r2tmp.id, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists r1tmp : sql2erc[EB2SQL(r_1), db] \cdot \\
&\quad \exists r2tmp : sql2erc[EB2SQL(r_2), db] \cdot \\
&\quad res_1 = id(r1tmp) \wedge res_2 = value(r2tmp) \wedge \\
&\quad value(r1tmp) = id(r2tmp)\}) \quad \text{sql2erc} \\
&= \{x \mapsto y \mid \exists z \cdot x \mapsto z \in rep_E(sql2erc[EB2SQL(r_1), db]) \wedge \\
&\quad z \mapsto y \in rep_E(sql2erc[EB2SQL(r_2), db])\} \quad \text{subs., } rep_E \\
&= \{x \mapsto y \mid \exists z \cdot x \mapsto z \in eb[r_1, rep(db)] \wedge \\
&\quad z \mapsto y \in eb[r_2, rep(db)]\} \quad \text{I.H.} \\
&= eb[r_1; r_2, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Case 24. $rep_E(sql2erc[EB2SQL(r_1 \circ r_2), db]) = eb[r_1 \circ r_2, rep(db)]$

Proof. The proof is by Case 23. \square

Case 25. $rep_E(sql2erc[EB2SQL(r^{-1}), db]) = eb[r^{-1}, rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(r^{-1}), db]) \\
&= rep_E(sql2erc[select rtmp.value, rtmp.id \\
&\quad from EB2SQL(r) rtmp, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad res_1 = sql2erc[rtmp.value, db] \wedge \\
&\quad \quad res_2 = sql2erc[rtmp.id, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad res_1 = value(rtmp) \wedge res_2 = id(rtmp)\}) \quad \text{sql2erc} \\
&= \{y \mapsto x \mid x \mapsto y \in rep_E(sql2erc[EB2SQL(r), db])\} \quad \text{subs., } rep_E \\
&= \{y \mapsto x \mid x \mapsto y \in eb[r, rep(db)]\} \quad \text{I.H.} \\
&= eb[r^{-1}, rep(db)] \quad \text{eb } \square
\end{aligned}$$

Case 26. $rep_E(sql2erc[EB2SQL(r[s]), db]) = eb[r[s], rep(db)]$

Proof.

$$\begin{aligned}
& rep_E(sql2erc[EB2SQL(r[s]), db]) \\
&= rep_E(sql2erc[select distinct rtmp.value \\
&\quad from EB2SQL(r) rtmp, EB2SQL(s) stmp \\
&\quad where rtmp.id = stmp.refkey, db]) \quad \text{EB2SQL} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad \exists stmp : sql2erc[EB2SQL(s), db] \cdot \\
&\quad \quad \quad res_1 = sql2erc[rtmp.value, db] \wedge \\
&\quad \quad \quad sql2erc[rtmp.id = stmp.refkey, db]\}) \quad \text{sql2erc} \\
&= rep_E(\{res_1 \mid res_1 : d_1 \# \wedge \\
&\quad \exists rtmp : sql2erc[EB2SQL(r), db] \cdot \\
&\quad \quad \exists stmp : sql2erc[EB2SQL(s), db] \cdot \\
&\quad \quad \quad res_1 = value(rtmp) \wedge id(rtmp) = refkey(stmp)\}) \quad \text{sql2erc} \\
&= \{y \mid \exists x \cdot x \in rep_E(sql2erc[EB2SQL(s), db]) \wedge \\
&\quad \quad x \mapsto y \in rep_E(sql2erc[EB2SQL(r), db])\} \quad \text{subs., } rep_E \\
&= \{y \mid \exists x \cdot x \in eb[s, rep(db)] \wedge \\
&\quad \quad x \mapsto y \in eb[r, rep(db)]\} \quad \text{I.H.} \\
&= eb[r[s], rep(db)] \quad \text{eb } \square
\end{aligned}$$

3.2 Proof of Soundness: Translating Multiple Assignments

Theorem 2. *For every set of Event-B assignment statements AS such that each assignment $A \in AS$ is translated directly to one or more SQL `insert` and `delete` statements by the EventB2SQL tool and every correctly typed database state db :*

$$rep(EB2SQL_{RES} \llbracket AS, db \rrbracket) = eb_{AS} \llbracket AS, rep(db) \rrbracket$$

That is, executing the database update statements generated by EventB2SQL from a set of Event-B assignment statements produces a database state that is equivalent to the Event-B model state produced by evaluating those assignment statements. The eb_{AS} operator was defined in Figure 3, and $EB2SQL_{RES}$ in Figure 5.

The proof proceeds by induction over the set of assignment statements. In the basis, $rep(EB2SQL_{RES} \llbracket \lambda, db \rrbracket) = rep(EB2SQL_{AS} \llbracket \lambda, EB2SQL_{OS} \llbracket \lambda, db \rrbracket \rrbracket) = rep(db) = eb_{AS} \llbracket \lambda, rep(db) \rrbracket$. The inductive step is again done by case analysis, with one case for each rule for $EB2SQL_A$ in Figure 5.

Case 1.

$$rep(EB2SQL_{RES} \llbracket s := s \cup s_1 \mid AS, db \rrbracket) = eb_{AS} \llbracket s := s \cup s_1 \mid AS, rep(db) \rrbracket$$

Proof.

$$\begin{aligned}
& rep(EB2SQL_{RES} \llbracket s := s \cup s_1 \mid AS, db \rrbracket) \\
&= rep(EB2SQL_{AS} \llbracket s := s \cup s_1 \mid AS, EB2SQL_{OS} \llbracket s := s \cup s_1 \mid AS, db \rrbracket \rrbracket) \quad EB2SQL_{RES} \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] sql2erc_A \llbracket \\
&\quad EB2SQL_A(s := s \cup s_1), \quad EB2SQL_{AS} \\
&\quad EB2SQL_{OS} \llbracket AS, EB2SQL_O \llbracket s := s \cup s_1, db \rrbracket \rrbracket \rrbracket \rrbracket) \quad EB2SQL_{OS} \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] sql2erc_A \llbracket \\
&\quad insert ignore into s select stmp.refkey from s' stmp, \quad EB2SQL_A \\
&\quad EB2SQL_{OS} \llbracket AS, [s' \rightarrow sql2erc \llbracket EB2SQL(s_1), db \rrbracket] db \rrbracket \rrbracket \rrbracket) \quad EB2SQL_O \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \cup sql2erc \llbracket \\
&\quad select stmp.refkey from s' stmp, db' \rrbracket] db' \rrbracket) \quad sql2erc_A, \dagger \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \cup \\
&\quad \{res_1 \mid res_1 : d_1 \# \wedge \exists stmp : sql2erc \llbracket s', db' \rrbracket \cdot \\
&\quad res_1 = sql2erc \llbracket stmp.refkey, db' \rrbracket \} \rrbracket] db' \rrbracket) \quad sql2erc \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \cup \\
&\quad \{refkey(stmp) \mid stmp \in sql2erc \llbracket EB2SQL(s_1), db \rrbracket \} \rrbracket] db' \rrbracket) \quad subs. \\
&= rep(EB2SQL_{AS} \llbracket AS, EB2SQL_{OS} \llbracket AS, [s \rightarrow db(s) \cup \\
&\quad \{refkey(stmp) \mid stmp \in sql2erc \llbracket EB2SQL(s_1), db \rrbracket \} \rrbracket] db \rrbracket \rrbracket) \quad *
\end{aligned}$$

$$\begin{aligned}
&= \text{eb}_{AS} \llbracket AS, \text{rep}([s \rightarrow db(s) \cup \text{sql2erc} \llbracket \text{EB2SQL}(E), db \rrbracket] db) \rrbracket && \text{I.H., subs.} \\
&= \text{eb}_{AS} \llbracket AS, [s \rightarrow \text{rep}_E(db(s) \cup \text{sql2erc} \llbracket \text{EB2SQL}(E), db \rrbracket)] \text{rep}(db) \rrbracket && \text{rep} \\
&= \text{eb}_{AS} \llbracket AS, [s \rightarrow \text{rep}_E(db(s)) \cup \text{rep}_E(\text{sql2erc} \llbracket \text{EB2SQL}(E), db \rrbracket)] \text{rep}(db) \rrbracket && \text{rep}_E \\
&= \text{eb}_{AS} \llbracket AS, [s \rightarrow \text{rep}(db)(s) \cup \text{rep}_E(\text{sql2erc} \llbracket \text{EB2SQL}(E), db \rrbracket)] \text{rep}(db) \rrbracket && \text{rep}_E \\
&= \text{eb}_{AS} \llbracket AS, [s \rightarrow \text{eb} \llbracket s, \text{rep}(db) \rrbracket \cup \text{eb} \llbracket E, \text{rep}(db) \rrbracket] \text{rep}(db) \rrbracket && \text{eb, Theorem 1} \\
&= \text{eb}_{AS} \llbracket AS, [s \rightarrow \text{eb} \llbracket s \cup E, \text{rep}(db) \rrbracket] \text{rep}(db) \rrbracket && \text{eb} \\
&= \text{eb}_{AS} \llbracket s := s \cup E \mid AS, \text{rep}(db) \rrbracket && \text{eb}_{AS} \square
\end{aligned}$$

In step †, define $db' = \text{EB2SQL}_{OS} \llbracket AS, [s' \rightarrow \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket] db \rrbracket$. In step *, note that:

- because an Event-B event can assign a machine variable at most once, no statement in AS assigns to s
- no assignment statement in AS refers to s'
- $db'(s) = db(s)$
- the critical parts of the right hand side of each assignment statement are evaluated by EB2SQL_O in the initial database state and the results stored in the state returned by EB2SQL_{OS} , bound to a primed version of the identifier on the left hand side of the assignment. These values are then used by the SQL **insert** and **delete** statements generated for each assignment by EB2SQL_A . The net effect is to evaluate the right hand side of each assignment in the initial database state, preserving the simultaneous assignment semantics of Event-B.

Case 2.

$$\text{rep}(\text{EB2SQL}_{RES} \llbracket s := s \setminus s_1 \mid AS, db \rrbracket) = \text{eb}_{AS} \llbracket s := s \setminus s_1 \mid AS, \text{rep}(db) \rrbracket$$

Proof.

$$\begin{aligned}
&\text{rep}(\text{EB2SQL}_{RES} \llbracket s := s \setminus s_1 \mid AS, db \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket s := s \setminus s_1 \mid AS, \text{EB2SQL}_{OS} \llbracket s := s \setminus s_1 \mid AS, db \rrbracket \rrbracket) && \text{EB2SQL}_{RES} \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A \llbracket \\
&\quad \text{EB2SQL}_A(s := s \setminus s_1), && \text{EB2SQL}_{AS} \\
&\quad \text{EB2SQL}_{OS} \llbracket AS, \text{EB2SQL}_O \llbracket s := s \setminus s_1, db \rrbracket \rrbracket \rrbracket) && \text{EB2SQL}_{OS} \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A \llbracket \\
&\quad \text{delete from } s \text{ where } s.\text{refkey in} \\
&\quad (\text{select } s1\text{tmp.refkey from } s' \text{ } s1\text{tmp}), && \text{EB2SQL}_A \\
&\quad \text{EB2SQL}_{OS} \llbracket AS, [s' \rightarrow \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket] db \rrbracket \rrbracket \rrbracket) && \text{EB2SQL}_O
\end{aligned}$$

$$\begin{aligned}
&= rep(EB2SQL_{AS}[[AS, [s' \rightarrow undef][s \rightarrow db'(s) \setminus sql2erc[\\
&\quad \text{select stmp.refkey from s stmp where stmp.refkey in} \\
&\quad \quad (\text{select s1tmp.refkey from s' s1tmp}), db']]]db']) \quad sql2erc_A, \dagger \\
&= rep(EB2SQL_{AS}[[AS, [s' \rightarrow undef][s \rightarrow db'(s) \setminus \\
&\quad \{res_1 \mid res_1 : d_1 \# \wedge \exists stmp : sql2erc[[s, db']]. \\
&\quad \quad sql2erc[[res_1 = stmp.refkey, db'] \wedge sql2erc[stmp.refkey \text{ in} \\
&\quad \quad \quad (\text{select s1tmp.refkey from s' s1tmp}), db']]]db']) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [s' \rightarrow undef][s \rightarrow db'(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db'(s) \wedge \exists s1tmp : sql2erc[[s', db']]. \\
&\quad \quad sql2erc[stmp.refkey = s1tmp.refkey, db']]]db']) \quad \text{subs.}, \\
&\quad \quad \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [s' \rightarrow undef][s \rightarrow db'(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db'(s) \wedge \exists s1tmp : db'(s') \cdot \\
&\quad \quad \quad refkey(stmp) = refkey(s1tmp)\}}]db']) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [s' \rightarrow undef][s \rightarrow db'(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db'(s) \wedge \\
&\quad \quad \quad stmp : sql2erc[[EB2SQL(s_1), db']]]db']) \quad \text{subs.} \\
&= rep(EB2SQL_{AS}[[AS, EB2SQL_{OS}[[AS, [s \rightarrow db(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db(s) \wedge stmp : sql2erc[[EB2SQL(s_1), db']]]db]]]] \quad * \\
&= eb_{AS}[[AS, rep([s \rightarrow db(s) \setminus (db(s) \cap sql2erc[[EB2SQL(s_1), db']]]db)]] \quad \text{I.H., subs.} \\
&= eb_{AS}[[AS, [s \rightarrow rep_E(db(s) \setminus (db(s) \cap sql2erc[[EB2SQL(s_1), db']]))rep(db)]] \quad rep \\
&= eb_{AS}[[AS, [s \rightarrow rep_E(db(s)) \setminus \\
&\quad (rep_E(db(s)) \cap rep_E(sql2erc[[EB2SQL(s_1), db']]))rep(db)]] \quad rep_E \\
&= eb_{AS}[[AS, [s \rightarrow rep(db)(s) \setminus \\
&\quad (rep(db)(s) \cap eb[s_1, rep(db)])]rep(db)]] \quad rep \\
&\quad \quad \quad \text{Theorem 1} \\
&= eb_{AS}[[AS, [s \rightarrow eb[s, rep(db)] \setminus \\
&\quad (eb[s, rep(db)] \cap eb[s_1, rep(db)])]rep(db)]] \quad eb \\
&= eb_{AS}[[AS, [s \rightarrow eb[s \setminus (s \cap s_1), rep(db)]rep(db)]] \quad eb \\
&= eb_{AS}[[AS, [s \rightarrow eb[s \setminus s_1, rep(db)]rep(db)]] \quad ** \\
&= eb_{AS}[[s := s \setminus s_1 \mid AS, rep(db)]] \quad eb_{AS} \square
\end{aligned}$$

In step \dagger , define $db' = EB2SQL_{OS}[[AS, [s' \rightarrow sql2erc[[EB2SQL(s_1), db]]db]]$.
In step $*$, see the justification for step $*$ in Case 1. In step $**$, note that $s_1 \setminus (s_1 \cap s_2) = s_1 \setminus s_2$.

Case 3.

$$rep(EB2SQL_{RES}[[s := s \cap s_1 \mid AS, db]]) = eb_{AS}[[s := s \setminus s_1 \mid AS, rep(db)]]$$

Proof.

$$\begin{aligned}
& rep(EB2SQL_{RES} \llbracket s := s \cap s_1 \rrbracket AS, db \rrbracket) \\
&= rep(EB2SQL_{AS} \llbracket s := s \cap s_1 \rrbracket AS, EB2SQL_{OS} \llbracket s := s \cap s_1 \rrbracket AS, db \rrbracket) \quad EB2SQL_{RES} \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] sql2erc_A \llbracket \\
&\quad EB2SQL_A(s := s \cap s_1), \quad EB2SQL_{AS} \\
&\quad EB2SQL_{OS} \llbracket AS, EB2SQL_O \llbracket s := s \cap s_1, db \rrbracket \rrbracket \rrbracket) \quad EB2SQL_{OS} \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] sql2erc_A \llbracket \\
&\quad delete from s where s.refkey in \\
&\quad (select sltmp.refkey from s' sltmp), \quad EB2SQL_A \\
&\quad EB2SQL_{OS} \llbracket AS, [s' \rightarrow sql2erc \llbracket EB2SQL(s \setminus s_1), db \rrbracket] db \rrbracket \rrbracket) \quad EB2SQL_O \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \setminus sql2erc \llbracket \\
&\quad select stmp.refkey from s stmp where stmp.refkey in \\
&\quad (select sltmp.refkey from s' sltmp), db' \rrbracket] db' \rrbracket) \quad sql2erc_A, \dagger \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \setminus \\
&\quad \{res_1 \mid res_1 : d_1 \# \wedge \exists stmp : sql2erc \llbracket s, db' \rrbracket. \\
&\quad sql2erc \llbracket res_1 = stmp.refkey, db' \rrbracket \wedge sql2erc \llbracket stmp.refkey in \\
&\quad (select sltmp.refkey from s' sltmp), db' \rrbracket \rrbracket] db' \rrbracket) \quad sql2erc \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db'(s) \wedge \exists sltmp : sql2erc \llbracket s', db' \rrbracket. \\
&\quad sql2erc \llbracket stmp.refkey = sltmp.refkey, db' \rrbracket \rrbracket] db' \rrbracket) \quad subs., \\
&\quad sql2erc \\
&= rep(EB2SQL_{AS} \llbracket AS, [s' \rightarrow undef] [s \rightarrow db'(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db'(s) \wedge stmp : sql2erc \llbracket s \setminus s_1, db' \rrbracket \rrbracket] db' \rrbracket) \quad subs., \\
&\quad sql2erc \\
&= rep(EB2SQL_{AS} \llbracket AS, [s \rightarrow db(s) \setminus \\
&\quad \{refkey(stmp) \mid stmp : db(s) \wedge stmp : sql2erc \llbracket s \setminus s_1, db \rrbracket \rrbracket] db \rrbracket) \quad * \\
&= eb_{AS} \llbracket AS, rep([s \rightarrow db(s) \setminus (db(s) \cap sql2erc \llbracket s \setminus s_1, db \rrbracket)] db) \rrbracket \quad I.H., subs. \\
&= eb_{AS} \llbracket AS, [s \rightarrow rep_E(db(s) \setminus (db(s) \cap sql2erc \llbracket s \setminus s_1, db \rrbracket))] rep(db) \rrbracket \quad rep \\
&= eb_{AS} \llbracket AS, [s \rightarrow rep_E(db(s)) \setminus \\
&\quad rep_E(db(s)) \cap rep_E(sql2erc \llbracket s \setminus s_1, db \rrbracket)] rep(db) \rrbracket \quad rep_E \\
&= eb_{AS} \llbracket AS, [s \rightarrow rep(db)(s) \setminus \\
&\quad rep(db)(s) \cap eb \llbracket s \setminus s_1, rep(db) \rrbracket] rep(db) \rrbracket \quad rep, \\
&\quad Theorem 1 \\
&= eb_{AS} \llbracket AS, [s \rightarrow eb \llbracket s \setminus (s \cap (s \setminus s_1)), rep(db) \rrbracket] rep(db) \rrbracket \quad eb \\
&= eb_{AS} \llbracket AS, [s \rightarrow eb \llbracket s \setminus (s \setminus s_1), rep(db) \rrbracket] rep(db) \rrbracket \quad ** \\
&= eb_{AS} \llbracket AS, [s \rightarrow eb \llbracket s \cap s_1, rep(db) \rrbracket] rep(db) \rrbracket \quad ** \\
&= eb_{AS} \llbracket s := s \cap s_1 \rrbracket AS, rep(db) \rrbracket \quad eb_{AS} \quad \square
\end{aligned}$$

In step \dagger , define $db' = EB2SQL_{OS} \llbracket AS, [s' \rightarrow sql2erc \llbracket EB2SQL(s \setminus s_1), db \rrbracket] db \rrbracket$.

In step *, see the justification for step * in Case 1. The steps marked ** are by simple set identities.

Case 4.

$$rep(EB2SQL_{RES}[[r := r \Leftarrow r_1 || AS, db]]) = eb_{AS}[[r := r \Leftarrow r_1 || AS, rep(db)]]$$

Proof.

$$\begin{aligned}
& rep(EB2SQL_{RES}[[r := r \Leftarrow r_1 || AS, db]]) \\
&= rep(EB2SQL_{AS}[[r := r \Leftarrow r_1 || AS, EB2SQL_{OS}[[r := r \Leftarrow r_1 || AS, db]]]]) \quad EB2SQL_{RES} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad EB2SQL_A(r := r \Leftarrow r_1), \quad EB2SQL_{AS} \\
&\quad EB2SQL_{OS}[[AS, EB2SQL_O[[r := r \Leftarrow r_1, db]]]]]]) \quad EB2SQL_{OS} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad delete from r where r.id in \\
&\quad (select r1tmp.id from r' r1tmp); \\
&\quad insert ignore into r \\
&\quad \quad select r2tmp.id, r2tmp.value from r' r2tmp;, \quad EB2SQL_A \\
&\quad EB2SQL_{OS}[[AS, [r' \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]]]) \quad sql2erc_O \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert ignore into r \\
&\quad \quad select r2tmp.id, r2tmp.value from r' r2tmp, \\
&\quad sql2erc_A[[delete from r where r.id in \\
&\quad \quad (select r1tmp.id from r' r1tmp), \\
&\quad EB2SQL_{OS}[[AS, [r' \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]]]) \quad sql2erc_A \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert ignore into r \\
&\quad \quad select r2tmp.id, r2tmp.value from r' r2tmp, \\
&\quad [r \rightarrow db(r) \setminus sql2erc[[select rtmp.id, rtmp.value from r rtmp \\
&\quad where rtmp.id in \\
&\quad \quad (select r1tmp.id from r' r1tmp), db']]db']]]) \quad sql2erc_A, \dagger
\end{aligned}$$

$$\begin{aligned}
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert\ ignore\ into\ r \\
&\quad\quad select\ r2tmp.id, r2tmp.value\ from\ r'\ r2tmp, \\
&\quad [r \rightarrow db(r) \setminus \{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \wedge \\
&\quad\quad \exists rtmp : sql2erc[[r, db'] \cdot sql2erc[[res_1 = rtmp.id, db'] \wedge \\
&\quad\quad\quad sql2erc[[res_2 = rtmp.value, db'] \wedge sql2erc[[rtmp.id\ in \\
&\quad\quad\quad (select\ r1tmp.id\ from\ r'\ r1tmp), db']\}]]db']]]) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert\ ignore\ into\ r \\
&\quad\quad select\ r2tmp.id, r2tmp.value\ from\ r'\ r2tmp, \\
&\quad [r \rightarrow db(r) \setminus \{id(rtmp), value(rtmp) \mid rtmp : db'(r) \wedge \\
&\quad\quad \exists r1tmp : sql2erc[[r', db'] \cdot sql2erc[[rtmp.id = r1tmp.id, db']\}]]db']]]) \quad \begin{array}{l} \text{subs.}, \\ sql2erc \end{array} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert\ ignore\ into\ r \\
&\quad\quad select\ r2tmp.id, r2tmp.value\ from\ r'\ r2tmp, \\
&\quad [r \rightarrow db(r) \setminus \{id(rtmp), value(rtmp) \mid rtmp : db'(r) \wedge \\
&\quad\quad \exists r1tmp : db'(r') \cdot id(rtmp) = id(r1tmp)\}]]db']]]) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad insert\ ignore\ into\ r \\
&\quad\quad select\ r2tmp.id, r2tmp.value\ from\ r'\ r2tmp, \\
&\quad [r \rightarrow db(r) \setminus \{rtmp \mid rtmp : db(r) \wedge \\
&\quad\quad id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\}]]db']]]) \quad \text{subs.} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef] \\
&\quad [r \rightarrow db''(r) \cup sql2erc[[\\
&\quad\quad select\ r2tmp.id, r2tmp.value\ from\ r'\ r2tmp, db'']]db'']] \quad \ddagger, sql2erc_A \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef] \\
&\quad [r \rightarrow db''(r) \cup \{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \\
&\quad\quad \exists r2tmp : sql2erc[[r', db'']] \cdot sql2erc[[res_1 = r2tmp.id, db'']] \wedge \\
&\quad\quad\quad sql2erc[[res_2 = r2tmp.value, db'']]db'']] \quad \begin{array}{l} sql2erc \\ \text{subs.}, \\ sql2erc \end{array} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef] \\
&\quad [r \rightarrow db''(r) \cup \{id(r2tmp), value(r2tmp) \mid r2tmp : db''(r')\}]]db'']] \quad \text{subs.} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef][r \rightarrow db''(r) \cup \{id(r2tmp), value(r2tmp) \mid \\
&\quad\quad r2tmp : sql2erc[[EB2SQL(r_1), db]]\}]]db'']] \quad \text{subs.} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef][r \rightarrow (db(r) \setminus \\
&\quad\quad \{rtmp \mid rtmp : db(r) \wedge id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\}) \\
&\quad\quad \cup sql2erc[[EB2SQL(r_1), db]]db'']] \quad \text{subs.}
\end{aligned}$$

$$\begin{aligned}
&= rep(EB2SQL_{AS}[[AS, [r \rightarrow (db(r) \setminus \\
&\quad \{rtmp \mid rtmp : db(r) \wedge id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\} \\
&\quad \cup sql2erc[[EB2SQL(r_1), db]]db]]) \quad * \\
&= eb_{AS}[[AS, rep([r \rightarrow (db(r) \setminus \\
&\quad \{rtmp \mid rtmp : db(r) \wedge id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\} \\
&\quad \cup sql2erc[[EB2SQL(r_1), db]]db]]) \quad \text{I.H.} \\
&= eb_{AS}[[AS, [r \rightarrow (rep_E(db(r)) \setminus \\
&\quad rep_E(\{rtmp \mid rtmp : db(r) \wedge id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\}) \\
&\quad \cup rep_E(sql2erc[[EB2SQL(r_1), db]])rep(db)]] \quad rep \\
&= eb_{AS}[[AS, [r \rightarrow (rep(db)(r) \setminus \\
&\quad \{x \mapsto y \mid x \mapsto y \in rep_E(db(r)) \wedge x \in rep_E(sql2erc[[EB2SQL(dom(r_1)), db]]\}) \\
&\quad \cup rep_E(sql2erc[[EB2SQL(r_1), db]])rep(db)]] \quad \text{subs.,} \\
&\quad \quad \quad rep \\
&= eb_{AS}[[AS, [r \rightarrow (eb[[r, rep(db)]] \setminus \\
&\quad \{x \mapsto y \mid x \mapsto y \in eb[[r, rep(db)]] \wedge x \in eb[[dom(r_1), rep(db)]]\}) \\
&\quad \cup eb[[r_1, rep(db)]]rep(db)]] \quad \text{Theorem 1,} \\
&\quad \quad \quad rep \\
&= eb_{AS}[[AS, [r \rightarrow (eb[[r, rep(db)]] \setminus \\
&\quad eb[[dom(r_1), rep(db)]] \triangleleft eb[[r, rep(db)]] \cup eb[[r_1, rep(db)]]rep(db)]] \quad eb \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r \setminus (dom(r_1) \triangleleft r) \cup r_1, rep(db)]]rep(db)]] \quad eb \\
&= eb_{AS}[[AS, [r \rightarrow eb[(dom(r_1) \triangleleft r) \cup r_1, rep(db)]]rep(db)]] \quad ** \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r \triangleleft r_1, rep(db)]]rep(db)]] \quad eb \\
&= eb_{AS}[[r := r \triangleleft r_1 \mid AS, rep(db)]] \quad eb_{AS} \square
\end{aligned}$$

In step †, define $db' = EB2SQL_{OS}[[AS, [r' \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]$. In step ‡, define $db'' = [r \rightarrow db(r) \setminus \{rtmp \mid rtmp : db(r) \wedge id(rtmp) \in sql2erc[[EB2SQL(r_1), db]]\}]db'$. In step *, see the justification for step * in Case 1. In step **, $r \setminus (s \triangleleft r) = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin s \triangleleft r\} = \{x \mapsto y \mid x \mapsto y \in r \wedge (x \mapsto y \notin r \vee x \notin s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge x \notin s\} = s \triangleleft r$.

Case 5.

$$rep(EB2SQL_{RES}[[r := s_1 \triangleleft r \mid AS, db]]) = eb_{AS}[[r := s_1 \triangleleft r \mid AS, rep(db)]]$$

Proof.

$$\begin{aligned}
&rep(EB2SQL_{RES}[[r := s_1 \triangleleft r \mid AS, db]]) \\
&= rep(EB2SQL_{AS}[[r := s_1 \triangleleft r \mid AS, EB2SQL_{OS}[[r := s_1 \triangleleft r \mid AS, db]]]]) \quad EB2SQL_{RES} \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef]sql2erc_A[[\\
&\quad EB2SQL_A(r := s_1 \triangleleft r), \quad EB2SQL_{AS} \\
&\quad EB2SQL_{OS}[[AS, EB2SQL_O[[r := s_1 \triangleleft r, db]]]]]]) \quad EB2SQL_{OS}
\end{aligned}$$

$$\begin{aligned}
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \text{sql2erc}_A \llbracket \\
&\quad \text{delete from r where r.id in} \\
&\quad (\text{select stmp.refkey from r' stmp}), \\
&\quad \text{EB2SQL}_{OS} \llbracket AS, [r' \rightarrow \text{sql2erc} \llbracket \text{EB2SQL}(s), db \rrbracket db \rrbracket \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \\
&\quad [r \rightarrow db(r) \setminus \text{sql2erc} \llbracket \text{select rtmp.id, rtmp.value from r rtmp} \\
&\quad \text{where rtmp.id in} \\
&\quad (\text{select stmp.refkey from r' stmp}), db' \rrbracket db' \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \\
&\quad [r \rightarrow db(r) \setminus \{ \text{res}_1, \text{res}_2 \mid \text{res}_1 : d_1 \# \wedge \text{res}_2 : d_2 \wedge \exists \text{rtmp} : \text{sql2erc} \llbracket r, db' \rrbracket \cdot \\
&\quad \text{sql2erc} \llbracket \text{res}_1 = \text{rtmp.id}, db' \rrbracket \wedge \text{sql2erc} \llbracket \text{res}_2 = \text{rtmp.value}, db' \rrbracket \wedge \\
&\quad \text{sql2erc} \llbracket \text{rtmp.id in (select stmp.refkey from r' stmp), db' \rrbracket} \rrbracket db' \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \\
&\quad [r \rightarrow db(r) \setminus \{ \text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \text{rtmp} : db'(r) \wedge \\
&\quad \exists \text{stmp} : \text{sql2erc} \llbracket r', db' \rrbracket \cdot \text{sql2erc} \llbracket \text{rtmp.id} = \text{stmp.refkey}, db' \rrbracket \rrbracket db' \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] [r \rightarrow db(r) \setminus \\
&\quad \{ \text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \text{rtmp} : db'(r) \wedge \text{id}(\text{rtmp}) : db'(r') \rrbracket db' \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] [r \rightarrow db(r) \setminus \{ \text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \\
&\quad \text{rtmp} : db(r) \wedge \text{id}(\text{rtmp}) : \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket \rrbracket db' \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r \rightarrow db(r) \setminus \{ \text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \\
&\quad \text{rtmp} : db(r) \wedge \text{id}(\text{rtmp}) : \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket \rrbracket db \rrbracket) \\
&= \text{eb}_{AS} \llbracket AS, \text{rep}([r \rightarrow db(r) \setminus \{ \text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \text{rtmp} : db(r) \wedge \\
&\quad \text{id}(\text{rtmp}) : \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket \rrbracket db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{rep}(db)(r) \setminus \{ x \mapsto y \mid \\
&\quad x \mapsto y \in \text{rep}(db)(r) \wedge x \in \text{rep}_E(\text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket) \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket \text{rep}(db)(r) \setminus \{ x \mapsto y \mid \\
&\quad x \mapsto y \in \text{rep}(db)(r) \wedge x \in \text{rep}_E(\text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket) \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r, \text{rep}(db) \rrbracket \rrbracket \setminus \{ x \mapsto y \mid \\
&\quad x \mapsto y \in \text{eb} \llbracket r, \text{rep}(db) \rrbracket \wedge x \in \text{eb} \llbracket s_1, \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r, \text{rep}(db) \rrbracket \rrbracket \setminus (\text{eb} \llbracket s_1, \text{rep}(db) \rrbracket \triangleleft \text{eb} \llbracket r, \text{rep}(db) \rrbracket) \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r \setminus (s \triangleleft r), \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket s_1 \triangleleft r, \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket \\
&= \text{eb}_{AS} \llbracket r := s_1 \triangleleft r \rrbracket \llbracket AS, \text{rep}(db) \rrbracket \rrbracket
\end{aligned}$$

Case 6.

$$\text{rep}(\text{EB2SQL}_{RES}[\![r := s_1 \triangleleft r \mid AS, db]\!]) = \text{eb}_{AS}[\![r := s_1 \triangleleft r \mid AS, \text{rep}(db)]\!]$$

Proof.

$$\begin{aligned}
& \text{rep}(\text{EB2SQL}_{RES}[\![r := s_1 \triangleleft r \mid AS, db]\!]) \\
& \text{rep}(\text{EB2SQL}_{AS}[\![r := s_1 \triangleleft r \mid AS, \text{EB2SQL}_{OS}[\![r := s_1 \triangleleft r \mid AS, db]\!]]]) \quad \text{EB2SQL}_{RES} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
& \quad \text{EB2SQL}_A(r := s_1 \triangleleft r), \quad \text{EB2SQL}_{AS} \\
& \quad \text{EB2SQL}_{OS}[\![AS, \text{EB2SQL}_O[\![r := s_1 \triangleleft r), db]\!]]]]]) \quad \text{EB2SQL}_{OS} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
& \quad \text{delete from } r \text{ where } r.\text{id in} \\
& \quad (\text{select stmp.refkey from } r' \text{ stmp}), \quad \text{EB2SQL}_A \\
& \quad \text{EB2SQL}_{OS}[\![AS, [r' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(\text{dom}(r) \setminus s_1), db]\!]]db]\!]]]) \quad \text{EB2SQL}_O \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
& \quad \text{sql2erc}[\![\text{select } rtmp.\text{id}, rtmp.\text{value from } r \text{ rtmp} \\
& \quad \text{where } rtmp.\text{id in} \\
& \quad (\text{select stmp.refkey from } r' \text{ stmp}), db']]\!]]]) \quad \text{sql2erc}_A, \dagger \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
& \quad \{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \exists rtmp : \text{sql2erc}[\![r, db']]\!]. \\
& \quad \text{sql2erc}[\![res_1 = rtmp.\text{id}, db']]\! \wedge \text{sql2erc}[\![res_2 = rtmp.\text{value}, db']]\! \wedge \\
& \quad \text{sql2erc}[\![rtmp.\text{id in} \\
& \quad (\text{select stmp.refkey from } r' \text{ stmp}), db']]\!]]]) \quad \text{sql2erc} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
& \quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db'(r) \wedge \\
& \quad \exists \text{stmp} : \text{sql2erc}[\![r', db']]\! \cdot \text{sql2erc}[\![rtmp.\text{id} = \text{stmp.refkey}, db']]\!]]db']]) \quad \text{subs.}, \text{sql2erc} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
& \quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db'(r) \wedge \\
& \quad \exists \text{stmp} : db'(r') \cdot \text{id}(rtmp) = \text{refkey}(\text{stmp})\}]db']]) \quad \text{sql2erc} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
& \quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \\
& \quad \text{id}(rtmp) : \text{sql2erc}[\![\text{EB2SQL}(\text{dom}(r) \setminus s_1), db]\!]]db']]) \quad \text{subs.} \\
& = \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r \rightarrow db(r) \setminus \{\text{id}(rtmp), \text{value}(rtmp) \mid \\
& \quad rtmp : db(r) \wedge \text{id}(rtmp) : \text{sql2erc}[\![\text{EB2SQL}(\text{dom}(r) \setminus s_1), db]\!]]db']]) \quad * \\
& = \text{eb}_{AS}[\![AS, \text{rep}([r \rightarrow db(r) \setminus \{\text{id}(rtmp), \text{value}(rtmp) \mid \\
& \quad rtmp : db(r) \wedge \text{id}(rtmp) : \text{sql2erc}[\![\text{EB2SQL}(\text{dom}(r) \setminus s_1), db]\!]]db')]\!]]] \quad \text{I.H.}
\end{aligned}$$

$$\begin{aligned}
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{rep}(db)(r) \setminus \text{rep}_E(\{\text{id}(\text{rtmp}), \text{value}(\text{rtmp}) \mid \\
&\quad \text{rtmp} : db(r) \wedge \text{id}(\text{rtmp}) : \text{sql2erc} \llbracket \text{EB2SQL}(\text{dom}(r) \setminus s_1), db \rrbracket \} \rrbracket] \text{rep}(db) \rrbracket && \text{rep} \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{rep}(db)(r) \setminus \{x \mapsto y \mid x \mapsto y \in \text{rep}(db)(r) \wedge && \text{subs.}, \\
&\quad x \in \text{rep}_E(\text{sql2erc} \llbracket \text{EB2SQL}(\text{dom}(r) \setminus s_1), db \rrbracket \} \rrbracket] \text{rep}(db) \rrbracket && \text{rep} \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r, \text{rep}(db) \rrbracket \setminus \{x \mapsto y \mid x \mapsto y \in \text{eb} \llbracket r, \text{rep}(db) \rrbracket \wedge && \text{eb}, \\
&\quad x \in \text{eb} \llbracket \text{dom}(r) \setminus s_1, \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket && \text{Theorem 1} \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r, \text{rep}(db) \rrbracket \setminus \text{eb} \llbracket (\text{dom}(r) \setminus s_1) \triangleleft r, \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket && \text{eb} \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r \setminus ((\text{dom}(r) \setminus s_1) \triangleleft r), \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket && \text{eb} \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket r \setminus (s_1 \triangleleft r), \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket && ** \\
&= \text{eb}_{AS} \llbracket AS, [r \rightarrow \text{eb} \llbracket s_1 \triangleleft r, \text{rep}(db) \rrbracket \rrbracket] \text{rep}(db) \rrbracket && *** \\
&= \text{eb}_{AS} \llbracket r := s_1 \triangleleft r \mid AS, \text{rep}(db) \rrbracket && \text{eb}_{AS} \square
\end{aligned}$$

In step †, define $db' = \text{EB2SQL}_{OS} \llbracket AS, [r' \rightarrow \text{sql2erc} \llbracket \text{EB2SQL}(\text{dom}(r) \setminus s_1), db \rrbracket \rrbracket db \rrbracket$
In step *, see the justification for step * in Case 1. In step **, see the justification for step ** in Case 4. In step ***, $r \setminus (s \triangleleft r) = \{x \mapsto y \mid x \mapsto y \in r \wedge x \mapsto y \notin (s \triangleleft r)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge (x \mapsto y \notin r \vee x \in s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge x \in s\} = s \triangleleft r$

Case 7.

$$\text{rep}(\text{EB2SQL}_{RES} \llbracket r := r \triangleright s_1 \mid AS, db \rrbracket) = \text{eb}_{AS} \llbracket r := r \triangleright s_1 \mid AS, \text{rep}(db) \rrbracket$$

Proof.

$$\begin{aligned}
&\text{rep}(\text{EB2SQL}_{RES} \llbracket r := r \triangleright s_1 \mid AS, \rrbracket) \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket r := r \triangleright s_1 \mid AS, \text{EB2SQL}_{OS} \llbracket r := r \triangleright s_1 \mid AS, db \rrbracket \rrbracket) && \text{EB2SQL}_{RES} \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \text{sql2erc}_A \llbracket && \\
&\quad \text{EB2SQL}_A(r := r \triangleright s_1), && \text{EB2SQL}_{AS} \\
&\quad \text{EB2SQL}_{OS} \llbracket AS, \text{EB2SQL}_O \llbracket r := r \triangleright s_1, db \rrbracket \rrbracket \rrbracket \rrbracket) && \text{EB2SQL}_{OS} \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] \text{sql2erc}_A \llbracket && \\
&\quad \text{delete from } r \text{ where } r.\text{value in} && \\
&\quad (\text{select stmp.refkey from } r' \text{ stmp}), && \text{EB2SQL}_A \\
&\quad \text{EB2SQL}_{OS} \llbracket AS, [r' \rightarrow \text{sql2erc} \llbracket \text{EB2SQL}(s_1), db \rrbracket \rrbracket db \rrbracket \rrbracket \rrbracket) && \text{EB2SQL}_O \\
&= \text{rep}(\text{EB2SQL}_{AS} \llbracket AS, [r' \rightarrow \text{undef}] [r \rightarrow db(r) \setminus \text{sql2erc} \llbracket && \\
&\quad \text{select rtmp.id, rtmp.value from } r \text{ rtmp where rtmp.value in} && \\
&\quad (\text{select stmp.refkey from } r' \text{ stmp}), db' \rrbracket \rrbracket db \rrbracket \rrbracket) && \text{sql2erc}_A, \dagger
\end{aligned}$$

$$\begin{aligned}
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef] \\
&\quad [r \rightarrow db(r) \setminus \{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \exists rtmp : sql2erc[[r, db'] \cdot \\
&\quad \quad sql2erc[[res_1 = rtmp.id, db'] \wedge sql2erc[[res_2 = rtmp.value, db'] \wedge \\
&\quad \quad sql2erc[[rtmp.value \text{ in} \\
&\quad \quad \quad (select \text{ stmp.refkey from } r' \text{ stmp}), db']]]db']) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef] \\
&\quad [r \rightarrow db(r) \setminus \{id(rtmp), value(rtmp) \mid rtmp : db'(r) \wedge \quad subs., \\
&\quad \quad \exists stmp : sql2erc[[r', db']] \cdot sql2erc[[rtmp.value = stmp.refkey, db']]]db']) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef][r \rightarrow db(r) \setminus \quad subs., \\
&\quad \quad \{id(rtmp), value(rtmp) \mid rtmp : db'(r) \wedge value(rtmp) : db'(r')\}]db']) \quad sql2erc \\
&= rep(EB2SQL_{AS}[[AS, [r \rightarrow db(r) \setminus \{id(rtmp), value(rtmp) \mid \\
&\quad \quad rtmp : db(r) \wedge value(rtmp) : sql2erc[[EB2SQL(s_1), db]]\}]db']) \quad subs., * \\
&= eb_{AS}[[AS, rep([r \rightarrow db(r) \setminus \{id(rtmp), value(rtmp) \mid \\
&\quad \quad rtmp : db(r) \wedge value(rtmp) : sql2erc[[EB2SQL(s_1), db]]\}]db')] \quad I.H. \\
&= eb_{AS}[[AS, [r \rightarrow rep(db)(r) \setminus rep_E(\{id(rtmp), value(rtmp) \mid \\
&\quad \quad rtmp : db(r) \wedge value(rtmp) : sql2erc[[EB2SQL(s_1), db]]\})]rep(db)] \quad rep \\
&= eb_{AS}[[AS, [r \rightarrow rep(db)(r) \setminus \{x \mapsto y \mid \quad subs., \\
&\quad \quad x \mapsto y \in rep(db)(r) \wedge y \in rep_E(sql2erc[[EB2SQL(s_1), db]])\}]rep(db)] \quad rep \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r, rep(db)]] \setminus \{x \mapsto y \mid \quad eb, \\
&\quad \quad x \mapsto y \in eb[[r, rep(db)]] \wedge y \in eb[[s_1, rep(db)]]\}]rep(db)] \quad Theorem 1 \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r, rep(db)]] \setminus eb[[r \triangleright s_1, rep(db)]]]rep(db)] \quad eb \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r \setminus (r \triangleright s), rep(db)]]]rep(db)] \quad eb \\
&= eb_{AS}[[AS, [r \rightarrow eb[[r \triangleright s_1, rep(db)]]]rep(db)] \quad ** \\
&= eb_{AS}[[r := r \triangleright s_1 || AS, rep(db)] \quad eb_{AS} \square
\end{aligned}$$

In step †, define $db' = EB2SQL_{OS}[[AS, [r' \rightarrow sql2erc[[EB2SQL(s_1), db]]]db]]$. In step *, see the justification for step * in Case 1. In step **, note that $r \setminus (r \triangleright s) = \{x \mapsto y \mid x \mapsto y \in r \wedge x \mapsto y \notin (r \triangleright s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge (x \mapsto y \notin r \vee y \notin s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge y \notin s\} = r \triangleright s$.

Case 8.

$$rep(EB2SQL_{RES}[[r := r \triangleright s_1 || AS, db]]) = eb_{AS}[[r := r \triangleright s_1 || AS, rep(db)]]$$

Proof.

$$\begin{aligned}
&rep(EB2SQL_{RES}[[r := r \triangleright s_1 || AS, db]]) \\
&= rep(EB2SQL_{AS}[[r := r \triangleright s_1 || AS, EB2SQL_{OS}[[r := r \triangleright s_1 || AS, db]]]]) \quad EB2SQL_{RES}
\end{aligned}$$

$$\begin{aligned}
&= \text{rep}(\text{sql2erc}_{AS}[\text{EB2SQL}_{AS}(AS), [r' \rightarrow \text{undef}]\text{sql2erc}_A[\\
&\quad \text{EB2SQL}_A(r := r \triangleright s_1), \quad \text{EB2SQL}_{OS}[AS, \text{EB2SQL}_O[r := r \triangleright s_1, db]]]]]) \quad \text{EB2SQL}_{AS} \\
&\quad \text{EB2SQL}_{OS} \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r' \rightarrow \text{undef}]\text{sql2erc}_A[\\
&\quad \text{delete from } r \text{ where } r.\text{value in} \\
&\quad (\text{select stmp.refkey from } r' \text{ stmp}), \quad \text{EB2SQL}_{OS}[AS, [r' \rightarrow \text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1), db]]db]]]) \quad \text{EB2SQL}_A \\
&\quad \text{sql2erc}_O \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
&\quad \{res_1, res_2 \mid res_1 : d_1 \# \wedge res_2 : d_2 \# \wedge \exists rtmp : \text{sql2erc}[r, db'] \cdot \\
&\quad \text{sql2erc}[res_1 = rtmp.\text{id}, db'] \wedge \text{sql2erc}[res_2 = rtmp.\text{value}, db'] \wedge \\
&\quad \text{sql2erc}[rtmp.\text{value in} \\
&\quad (\text{select stmp.refkey from } r' \text{ stmp}), db']\}]]db']) \quad \text{sql2erc}_A, \dagger \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
&\quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db'(r) \wedge \exists \text{stmp} : \text{sql2erc}[r', db'] \cdot \\
&\quad \text{sql2erc}[rtmp.\text{value} = \text{stmp.refkey}, db']\}]]db']) \quad \text{subs.}, \\
&\quad \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
&\quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \exists \text{stmp} : db'(r') \cdot \\
&\quad \text{value}(rtmp) = \text{refkey}(\text{stmp})\}]]db']) \quad \text{subs.}, \\
&\quad \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r' \rightarrow \text{undef}][r \rightarrow db(r) \setminus \\
&\quad \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \exists \text{stmp} : \text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1, db)] \cdot \\
&\quad \text{value}(rtmp) = \text{refkey}(\text{stmp})\}]]db']) \quad \text{subs.} \\
&= \text{rep}(\text{EB2SQL}_{AS}[AS, [r \rightarrow db(r) \setminus \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \\
&\quad \exists \text{stmp} : \text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1), db] \cdot \text{value}(rtmp) = \text{refkey}(\text{stmp})\}]]db']) \quad * \\
&= \text{eb}_{AS}[AS, \text{rep}([r \rightarrow db(r) \setminus \{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \\
&\quad \exists \text{stmp} : \text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1), db] \cdot \text{value}(rtmp) = \text{refkey}(\text{stmp})\}]]db']) \quad \text{I.H.} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{rep}(db)(r) \setminus \text{rep}_E(\{\text{id}(rtmp), \text{value}(rtmp) \mid rtmp : db(r) \wedge \\
&\quad \exists \text{stmp} : \text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1), db] \cdot \text{value}(rtmp) = \text{refkey}(\text{stmp})\})]\text{rep}(db)]] \quad \text{rep} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{rep}(db)(r) \setminus \{x \mapsto y \mid x \mapsto y \in \text{rep}(db)(r) \wedge \\
&\quad y \in \text{rep}_E(\text{sql2erc}[\text{EB2SQL}(\text{ran}(r) \setminus s_1), db])\}]\text{rep}(db)]] \quad \text{subs.}, \\
&\quad \text{rep} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{eb}[r, \text{rep}(db)] \setminus \{x \mapsto y \mid x \mapsto y \in \text{eb}[r, \text{rep}(db)] \wedge \\
&\quad y \in \text{eb}[\text{ran}(r) \setminus s_1, \text{rep}(db)]\}]\text{rep}(db)]] \quad \text{eb}, \\
&\quad \text{Theorem 1} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{eb}[r, \text{rep}(db)] \setminus \text{eb}[r \triangleright (\text{ran}(r) \setminus s_1), \text{rep}(db)]]\text{rep}(db)]] \quad \text{eb} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{eb}[r \setminus (r \triangleright (\text{ran}(r) \setminus s_1)), \text{rep}(db)]]\text{rep}(db)]] \quad \text{eb} \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{eb}[r \setminus (r \triangleright s_1), \text{rep}(db)]]\text{rep}(db)]] \quad ** \\
&= \text{eb}_{AS}[AS, [r \rightarrow \text{eb}[r \triangleright s_1, \text{rep}(db)]]\text{rep}(db)]] \quad *** \\
&= \text{eb}_{AS}[r := r \triangleright s_1 \mid AS, \text{rep}(db)] \quad \text{eb}_{AS} \square
\end{aligned}$$

In step †, define $db' = \text{EB2SQL}_{OS}[\![AS, [r' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(\text{ran}(r) \setminus s_1), db]\!]]db]\!]$.
 In step *, see the justification for step * in Case 1. In step **, $r \triangleright (\text{ran}(r) \setminus s) = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in (\text{ran}(r) \setminus s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in \text{ran}(r) \wedge y \notin s\} = \{x \mapsto y \mid x \mapsto y \in r \wedge y \notin s\} = r \triangleright s$. In step ***, $r \setminus (r \triangleright s) = \{x \mapsto y \mid x \mapsto y \in r \wedge x \mapsto y \notin (r \triangleright s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge (x \mapsto y \notin r \vee y \in s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge (x \mapsto y \notin r \vee y \in s)\} = \{x \mapsto y \mid x \mapsto y \in r \wedge y \in s\} = r \triangleright s$.

Case 9.

$$\text{rep}(\text{EB2SQL}_{RES}[\![s := s_1 \parallel AS, db]\!]) = \text{eb}_{AS}[\![s := s_1 \parallel AS, \text{rep}(db)]\!]$$

Proof.

$$\begin{aligned}
& \text{rep}(\text{EB2SQL}_{RES}[\![s := s_1 \parallel AS, db]\!]) \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![s := s_1 \parallel AS, \text{EB2SQL}_{OS}[\![s := s_1 \parallel AS, db]\!]]]) && \text{EB2SQL}_{RES} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
&\quad \text{EB2SQL}_A(s := s_1), && \text{EB2SQL}_{AS} \\
&\quad \text{EB2SQL}_{OS}[\![AS, \text{EB2SQL}_O[\![s := s_1, db]\!]]]\!]]]) && \text{EB2SQL}_{OS} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
&\quad \text{delete from s;} \\
&\quad \text{insert ignore into s select sltmp.refkey from s' sltmp,} && \text{EB2SQL}_A \\
&\quad \text{EB2SQL}_{OS}[\![AS, [s' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db]\!]]]\!]]]) && \text{sql2erc}_O \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
&\quad \text{insert ignore into s select sltmp.refkey from s' sltmp,} \\
&\quad \text{sql2erc}_A[\![\text{delete from s,} \\
&\quad \text{EB2SQL}_{OS}[\![AS, [s' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db]\!]]]\!]]]) && \text{sql2erc}_A \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] \text{sql2erc}_A[\![\\
&\quad \text{insert ignore into s select sltmp.refkey from s' sltmp,} \\
&\quad [s \rightarrow \{\}] \text{EB2SQL}_{OS}[\![AS, [s' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db]\!]]]\!]]]) && \text{sql2erc}_A \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] [s \rightarrow db'(s) \cup \\
&\quad \text{sql2erc}[\![\text{select sltmp.refkey from s' sltmp, } db']]\!]]db']]) && \text{sql2erc}_A, \dagger \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] [s \rightarrow db'(s) \cup \\
&\quad \{\text{res}_1 \mid \text{res}_1 : d_1 \# \wedge \exists \text{sltmp} : \text{sql2erc}[\![s', db']]\!]. \\
&\quad \text{sql2erc}[\![\text{res}_1 = \text{sltmp.refkey, } db']]\!]]db']]) && \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] [s \rightarrow \{\} \cup \\
&\quad \{\text{refkey}(\text{sltmp}) \mid \text{sltmp} : db'(s')\}]\!]]db']]) && \text{subs.,} \\
&\quad \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] [s \rightarrow \\
&\quad \{\text{refkey}(\text{sltmp}) \mid \text{sltmp} : \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db']]\!]]]) && \text{subs.} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s' \rightarrow \text{undef}] [s \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db']]\!]]]) && \text{subs.} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [s \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]]db']]\!]]]) && *
\end{aligned}$$

$$\begin{aligned}
&= \text{eb}_{AS}[\![AS, \text{rep}([s \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]\text{db})]\!]] && \text{I.H.} \\
&= \text{eb}_{AS}[\![AS, [s \rightarrow \text{rep}_E(\text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!])]\!]\text{rep}(db)] && \text{rep} \\
&= \text{eb}_{AS}[\![AS, [s \rightarrow \text{eb}[s_1, \text{rep}(db)]]]\!]\text{rep}(db)] && \text{Theorem 1} \\
&= \text{eb}_{AS}[\![s := s_1 \parallel AS, \text{rep}(db)]\!] && \text{eb}_{AS} \square
\end{aligned}$$

In step †, define $db' = [s \rightarrow \{\}] \text{EB2SQL}_{OS}[\![AS, [s' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(s_1), db]\!]\text{db}]\!]$.
In step *, see the justification for step * in Case 1.

Case 10.

$$\text{rep}(\text{EB2SQL}_{RES}[\![r := r_1 \parallel AS, db]\!]) = \text{eb}_{AS}[\![r := r_1 \parallel AS, \text{rep}(db)]\!]$$

Proof.

$$\begin{aligned}
&\text{rep}(\text{EB2SQL}_{RES}[\![r := r_1 \parallel AS, db]\!]) \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![r := r_1 \parallel AS, \text{EB2SQL}_{OS}[\![r := r_1 \parallel AS, db]\!]]\!]) && \text{EB2SQL}_{RES} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!]\text{sql2erc}_A[\![\\
&\quad \text{EB2SQL}_A(r := r_1), && \text{EB2SQL}_A \\
&\quad \text{EB2SQL}_{OS}[\![AS, \text{EB2SQL}_O[\![r := r_1, db]\!]]]\!]]\!]) && \text{EB2SQL}_{OS} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!]\text{sql2erc}_A[\![\\
&\quad \text{delete from r;} \\
&\quad \text{insert ignore into r select r1tmp.id, r1tmp.value from r' r1tmp,} && \text{EB2SQL}_A \\
&\quad \text{EB2SQL}_{OS}[\![AS, [r' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(r_1), db]\!]\text{db}]\!]]]\!]]\!]) && \text{sql2erc}_O \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!]\text{sql2erc}_A[\![\\
&\quad \text{insert ignore into r select r1tmp.id, r1tmp.value from r' r1tmp,} \\
&\quad \text{sql2erc}_A[\![\text{delete from r,} \\
&\quad \text{EB2SQL}_{OS}[\![AS, [r' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(r_1), db]\!]\text{db}]\!]]]\!]]\!]) && \text{sql2erc}_A \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!]\text{sql2erc}_A[\![\\
&\quad \text{insert ignore into r select r1tmp.id, r1tmp.value from r' r1tmp,} \\
&\quad [r \rightarrow \{\}] \text{EB2SQL}_{OS}[\![AS, [r' \rightarrow \text{sql2erc}[\![\text{EB2SQL}(r_1), db]\!]\text{db}]\!]]]\!]]\!]) && \text{sql2erc}_A \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!][r \rightarrow db'(r) \cup \\
&\quad \text{sql2erc}[\![\text{select r1tmp.id, r1tmp.value from r' r1tmp, db']]\text{db'}]\!]) && \text{sql2erc}_A, \dagger \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!][r \rightarrow db'(r) \cup \\
&\quad \{\text{res}_1, \text{res}_2 \mid \text{res}_1 : d_1 \# \wedge \text{res}_2 : d_2 \# \wedge \exists r1tmp : \text{sql2erc}[\![r', db']]\cdot \\
&\quad \text{sql2erc}[\![\text{res}_1 = r1tmp.\text{id}, db']]\wedge \text{sql2erc}[\![\text{res}_2 = r1tmp.\text{value}, db']]\}\!]\text{db'}]\!]) && \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!][r \rightarrow \{\}] \cup && \text{subs.,} \\
&\quad \{\text{id}(r1tmp), \text{value}(r1tmp) \mid r1tmp : db'(r)\}\!]\text{db'}]\!]) && \text{sql2erc} \\
&= \text{rep}(\text{EB2SQL}_{AS}[\![AS, [r' \rightarrow \text{undef}]\!][r \rightarrow \\
&\quad \{\text{id}(r1tmp), \text{value}(r1tmp) \mid r1tmp : \text{sql2erc}[\![\text{EB2SQL}(r_1), db]\!]\}\!]\text{db'}]\!]) && \text{subs.}
\end{aligned}$$

$$\begin{aligned}
&= rep(EB2SQL_{AS}[[AS, [r' \rightarrow undef][r \rightarrow sql2erc[[EB2SQL(r_1), db]]db']]]) && \text{subs.} \\
&= rep(EB2SQL_{AS}[[AS, [r \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]) && * \\
&= eb_{AS}[[AS, rep([r \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]) && \text{I.H.} \\
&= eb_{AS}[[AS, [r \rightarrow rep_E(sql2erc[[EB2SQL(r_1), db]])rep(db)]] && rep \\
&= eb_{AS}[[AS, [r \rightarrow eb[r_1, rep(db)]]rep(db)]] && \text{Theorem 1} \\
&= eb_{AS}[[r := r_1 || AS, rep(db)]] && eb_{AS} \quad \square
\end{aligned}$$

In step \dagger , define $db' = [r \rightarrow \{\}]EB2SQL_{OS}[[AS, [r' \rightarrow sql2erc[[EB2SQL(r_1), db]]db]]$. In step $*$, see the justification for step $*$ in Case 1.

4 Related Work

Soundness proofs are important for any code generation tool. Proof techniques vary depending on language properties and semantics, and the availability of proof support tools. For example, the correctness of a translation from a large subset of C to assembly language was verified in the CompCert project [13]. All verified parts of the CompCert compiler are written in Coq [4], as are the semantics of the source and target languages. The soundness (semantic preservation) proof was completed with the aid of the Coq proof assistant. This approach is closely related to the idea of embedding a formal specification language in the language of an automated theorem prover, which is a popular technique for reasoning about soundness [5, 12].

The authors of [9] take a correctness-by-construction approach to verify that the code generated by their system is a refinement of the Event-B model it was generated from. In particular, their tool automatically produces the invariants needed to show that the generated code is a refinement, as well as the invariants needed for proving the well-definedness of arithmetic operations. These proofs can then be carried out with the aid of automated theorem provers. However, their tool can translate only a limited subset of Event-B with restricted schedules, and a separate correctness proof needs to be performed for each model that is translated. The E-SPARK approach [17] similarly translates an Event-B model to an implementation in the SPARK subset of Ada, annotated with the loop invariants and pre- and post-conditions needed for verification. The verification conditions can then be discharged with SPARK proof tools. E-SPARK translates only concrete, sequential Event-B models and all events are merged into a single procedure in the implementation.

The CLawZ toolset [18] also uses a correctness-by-construction approach. Here, a Simulink[®] model is translated to both a Z specification and executable code (in the SPARK subset of Ada). Additionally, the tool generates the verification conditions needed to show that the code is a refinement of the Z specification. The authors state that the Theorem Prover included in the toolset seems to be able to discharge these verification conditions without human interaction

due to the regular structure of the generated specifications and code. Note that Event-B provides a much richer modeling notation than Simulink®.

The EB³TG tool [10] generates database applications (implemented in Java) from EB³ specifications. EB³ is a trace-based formal specification language – system outputs are specified in terms of sequences of input events – and so is quite different from state-based languages such as Event-B. The EB³TG tool synthesizes a database transaction for each type of input event, as well as the SQL statements for creating database tables. The authors provide sketches of manual proofs of both the soundness and completeness of their code generation technique, using a pair of morphisms that map from EB³ traces to database states, and from database states back to traces. Both proofs proceed by structural induction, with the soundness proof using the morphism from traces to states to show the appropriate correspondence, and the completeness proof the morphism from states to traces. The soundness proof is similar to ours in that a mapping is used to show the correspondence of states after the evaluation of a specification construct and the execution of code generated from that construct. The translation performed by EventB2SQL is not complete, as there are Event-B constructs such as operations on infinite sets for which EventB2SQL does not generate code. As such, no proof of completeness is possible.

Perhaps the most closely related work to ours is the UB2SQL tool [14]. The authors propose a methodology for developing information systems that begins with UML class diagrams specifying the structure of the data in the system, and state and collaboration diagrams modeling the functionality. These diagrams are automatically translated to a number of abstract B [1] machines, which are then refined in a largely automatic manner to implementation-level B machines. Finally, statements for creating the necessary database tables and a Java/JDBC implementation of the operations are derived. The authors were working on automating the code generation step at the time the paper cited above was published. They state that proving the soundness of their code generation technique is difficult because code derivation is done outside of any formal environment, and indeed their code generation rules are presented somewhat informally. However, they have proven the soundness of the refinement rules used to product the implementation-level B machines using AtelierB [3], and have integrated tactics for each rule into the B prover. The key philosophical difference from our work is that UB2SQL is intended for integrating the formality of the B method into the development of information systems with an explicit database component, while EventB2SQL is meant for generating code from any Event-B model for which persistent state is useful.

By comparison with the approaches described above, our soundness proof is rather straightforward. This is due in part to the restricted scope of the problem we are considering. By looking only at the translation of sets of assignment statements, we avoid issues such as atomicity, scheduling, . . . that a full soundness proof would need to consider. Additionally, we were fortunate to find two formal semantics for SQL [11, 15] that were so compatible with the way that we had formalized our translation algorithm and the mathematical definitions of Event-B operations.

5 Future Work and Conclusion

The semantics and proof presented in this work consider only Event-B operators that EB2SQL translates directly to SQL queries. Several other constructs (powerset, direct product, parallel product, partition, set comprehensions and quantified assertions) are translated to a mixture of SQL and Java code. Defining the semantics of this part of the translation is considerably more challenging, as is proving its soundness. Defining a full semantics for the translation of events presents a similar challenge, as an event is translated to a Java method that contains JDBC API calls. Finally, a machine checked version of the proof presented in this paper would provide a substantially greater level of assurance of the soundness of the translation. Given the number of languages and translation operators involved, conducting such a proof presents a formidable challenge.

As noted in the introduction to this work, soundness is critical for any code generator for Event-B. Because EventB2SQL translates abstract machines and so avoids the refinement chains typically seen in Event-B developments, the model and implementation are very different and the correspondence between them is not readily apparent to the user. Hence, it is particularly important to provide assurance that the code generated by the EventB2SQL tool is sound. As the translation of sets of Event-B assignment statements presented and proven sound in this work forms the core of the tool, the proof provides that assurance, giving users confidence that the generated code satisfies the correctness and safety properties that were verified for the original Event-B model.

References

- [1] J. R. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
- [2] J.-R. Abrial. *Modeling in Event-B: System and Software Design*. Cambridge University Press, New York, NY, USA, 2010.
- [3] Atelier B. http://www.atelierb.eu/index_en.html.
- [4] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.
- [5] J. Bowen and M. Gordon. Z and HOL. In *Z User Workshop, Workshops in Computing*, pages 141–167, Cambridge, U.K., 1994. Springer.
- [6] N. Cataño and T. Wahls. A case study on code generation of an ERP system from Event-B. In *IEEE QRS 2015*, 2015.
- [7] A. Edmunds and M. Butler. Tool support for Event-B code generation. In *WS-TBFM2010*, Québec, Canada, 2010. John Wiley and Sons.

- [8] A. Edmunds and M. Butler. Tasking Event-B: An extension to Event-B for generating concurrent code. In *Programming Language Approaches to Concurrency and Communication-Centric Software PLACES*, Saarbrücken, Germany, 2011. Springer.
- [9] A. Fürst, T. S. Hoang, D. Basin, K. Desai, N. Sato, and K. Miyazaki. Code generation for Event-B. In *Integrated Formal Methods 2014*, volume 8737, 2014.
- [10] F. Gervais, M. Frappier, and R. Laleau. Generating relational database transactions from EB^3 attribute definitions. *Software and System Modeling*, 8(3):423 – 445, 2009.
- [11] M. Gogolla. Formal semantics of SQL. In *An Extended Entity-Relationship Model*, volume 767 of *LNCS*, pages 99–120. Springer, 1994.
- [12] M. Jacquél, K. Berkani, D. Delahaye, and C. Dubois. Verifying B proof rules using deep embedding and automated theorem proving. In *SEFM 2011*, pages 253–268. Springer, 2011.
- [13] X. Leroy. Formal verification of a realistic compiler. *CACM*, 52(7):107–115, 2009.
- [14] A. Mammar and R. Laleau. From a B formal specification to an executable code: Application to the relational database domain. *Inf. Softw. Technol.*, 48(4):253–279, Apr. 2006.
- [15] S. Meira, R. Motz, and F. Tepedino. A Formal Semantics for SQL. *International Journal of Computer Mathematics*, 34:43–63, 1990.
- [16] D. Méry and N. K. Singh. Automatic code generation from Event-B models. In *Proceedings of the Second SoICT*, SoICT ’11, Hanoi, Vietnam, 2011. ACM.
- [17] R. Murali and A. Ireland. E-SPARK: Automated generation of provably correct code from formally verified designs. In *AvoCS 2012*, 2012.
- [18] C. O’Halloran. Automated verification of code automatically generated from Simulink[®]. *Automated Software Engineering*, 20(2):237–264, 2013.
- [19] Rodin user’s handbook. http://handbook.event-b.org/current/html/mathematical_notation.html.
- [20] T. Wahls. MedicationChecker: Development of a formally verified android application with EventB2SQL, 2016. Submitted to IEEE QRS 2016.
- [21] Q. Wang and T. Wahls. Translating Event-B machines to database applications. In D. Giannakopoulou and G. Salaün, editors, *SEFM 2014*, volume 8702 of *LNCS*, pages 265–270. Springer, 2014.
- [22] S. Wright. Automatic generation of C from Event-B. In *Workshop on IM_FMT*, Nantes, France, 2009. Springer.